

Verteilte Anwendungen

*Hochschule Furtwangen
Prof. Dr. Dirk Eisenbiegler
Stand: Wintersemester 2021/2022*

Organisatorisches

- x Der Praktikumsschein und die Klausur zur Vorlesung sind zwei eigenständige Prüfungsleistungen, die Sie unabhängig voneinander erbringen müssen.
- x Für das Praktikum besteht Anwesenheitspflicht. Tragen Sie sich bitte unbedingt bei jedem Praktikumstermin in die Anwesenheitslisten ein!
- x Die Programmieraufgaben des Praktikums müssen erfolgreich bearbeitet und die Ergebnisse anschließend termingerecht abgegeben werden. Beachten Sie hierzu die Hinweise im Praktikum!

Übersicht

Web-Applications

Datenbankanbindung

REST-Services

Verteilte Anwendungen

Nebenläufigkeit

Socket-Programmierung

Vorlesung Programmierung

Kapitel 1

Nebenläufigkeit

Motivation

- x Die bisher betrachteten Programme
 - ⇒ richten sich an einen Ausführenden
 - ⇒ beschreiben eine Sequenz elementarer Befehle, die von dem Ausführenden abzuarbeiten sind
- x Allgemein bezeichnet man die gleichzeitige Ausführung mehrerer Programme als Nebenläufigkeit.
- x Java unterstützt Nebenläufigkeit.

Prozesse in Betriebssystemen

Moderne Betriebssysteme unterstützen Nebenläufigkeit in Form von Prozessen.

- x Die Prozesse werden vom Betriebssysteme verwaltet.
- x Jeder Prozess führt ein Programm aus.
- x Die Prozesse werden gleichzeitig ausgeführt.
- x Jeder Prozess bekommt einen Bereich im Hauptspeicher zugeteilt (logischer Adressraum). Der Prozess kann nur auf die Daten im eigenen Bereich zugreifen.
- x Die Prozesse können vom Betriebssystem Ressourcen anfordern, die sie nach der Zuteilung exklusiv nutzen können.
- x Zur Kommunikation und Synchronisation zwischen den Prozessen müssen explizit Mechanismen des Betriebssystems verwendet werden.

Softwarethreads

Einige Betriebssysteme unterstützen auch Softwarethreads.

- x Softwarethreads ähneln Prozessen.

 - ⇒ Sie werden vom Betriebssystem verwaltet.

 - ⇒ Sie werden gleichzeitig ausgeführt.

- x Unterschied: Eine Gruppe von Softwarethreads läuft in einem gemeinsamen logischen Adressraum.

- x Folgen:

 - ⇒ Auf die Daten im logischen Adressraum kann gemeinsam zugegriffen werden.

 - ⇒ Diese Zugriffe auf gemeinsam verwendete Daten müssen sinnvoll abgestimmt, synchronisiert werden. Nur so ist eine sinnvolle Zusammenarbeit zwischen den Softwarethreads möglich.

Technische Realisierung der Nebenläufigkeit

- x Fall 1: Der Computer hat nur einen einfachen Mikroprozessor (ohne Hardwarethreads).
 - ⇒ Es kann immer nur ein Programm gleichzeitig ausgeführt werden.
 - ⇒ Nebenläufigkeit durch Time-Slicing:
 - x Die Prozesse/Softwarethreads werden reihum jeweils ein Stückchen ausgeführt
 - x Das Umschalten von einem Prozess/Softwarethread auf den nächsten wird durch regelmäßige Interrupts ausgelöst.
- x Fall 2: Der Mikroprozessor unterstützt Hardwarethreads.
 - ⇒ Der Mikroprozessor kann eine begrenzten Anzahl Prozesse/Softwarethreads gleichzeitig ausführen (Überlappung der Ausführung in der Befehls-Pipeline)
 - ⇒ Kombination mit Time-Slicing

... Technische Realisierung der Nebenläufigkeit

- x Fall 3: Der Computer verfügt über mehrere Mikroprozessoren.
 - ⇒ Die Prozesse/Threads können auf unterschiedlichen Mikroprozessoren ausgeführt werden.
 - ⇒ Kombination mit Time-Slicing und Aufteilung auf Hardware-Threads (so vorhanden)

Multithreading in Java

- x Java verwaltet mehrere Threads.
 - ⇒ Thread (englisch) = Faden (deutsch)
 - ⇒ in diesem Zusammenhang: Handlungsfaden
- x Jeder Thread führt ein Stück Programmcode aus.
- x Alle Threads arbeiten gleichzeitig.
- x Threads in Java können auf die gleichen Daten zugreifen.
 - ⇒ Dies entspricht vom Konzept den Softwarethreads in Betriebssystemen.
- x Java verfügt über Mechanismen zur Synchronisation zwischen Threads.

Grundmechanismen

- x Die Programmiersprache Java bietet mehrere Grundmechanismen zum Arbeiten mit Threads an:
 - ⇒ Mechanismen zum Erzeugen und Beenden von Threads
 - ⇒ Mechanismen zur Kommunikation und Synchronisation zwischen Threads
- x Die Umsetzung dieser Grundmechanismen durch die Java Virtual Machine kann in unterschiedlichen Betriebssystemen und Rechnerarchitekturen unterschiedlich realisiert sein.
- x Aus Sicht der Programmierung müssen diese Unterschiede nicht beachtet werden:
 - ⇒ Einmal geschriebene Java-Programme, die das Java-Multithreading verwenden, werden auf unterschiedliche Rechnerarchitekturen und Betriebssystemen auf unterschiedliche Weise ausgeführt, ohne dass dazu der Programmcode geändert werden müsste.

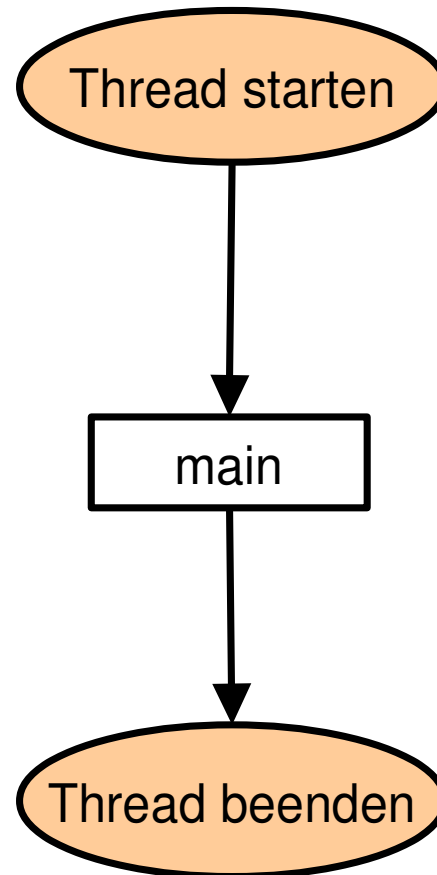
1.1 Das Starten und das Beenden von Threads

Ohne Multithreading

In den bisher betrachteten Programmen war immer nur ein Thread mit der Ausführung des Programmcodes befasst:

- x Wird ein Java-Programm gestartet, so erzeugt die Java Virtual Machine einen Thread.
- x Der Thread führt die main-Methode der angegebenen Klasse aus.
- x Sobald der Thread die main-Methode bearbeitet hat, wird der Thread beendet.
- x Die Ausführung des Programms ist damit beendet.

Programm mit einem Thread

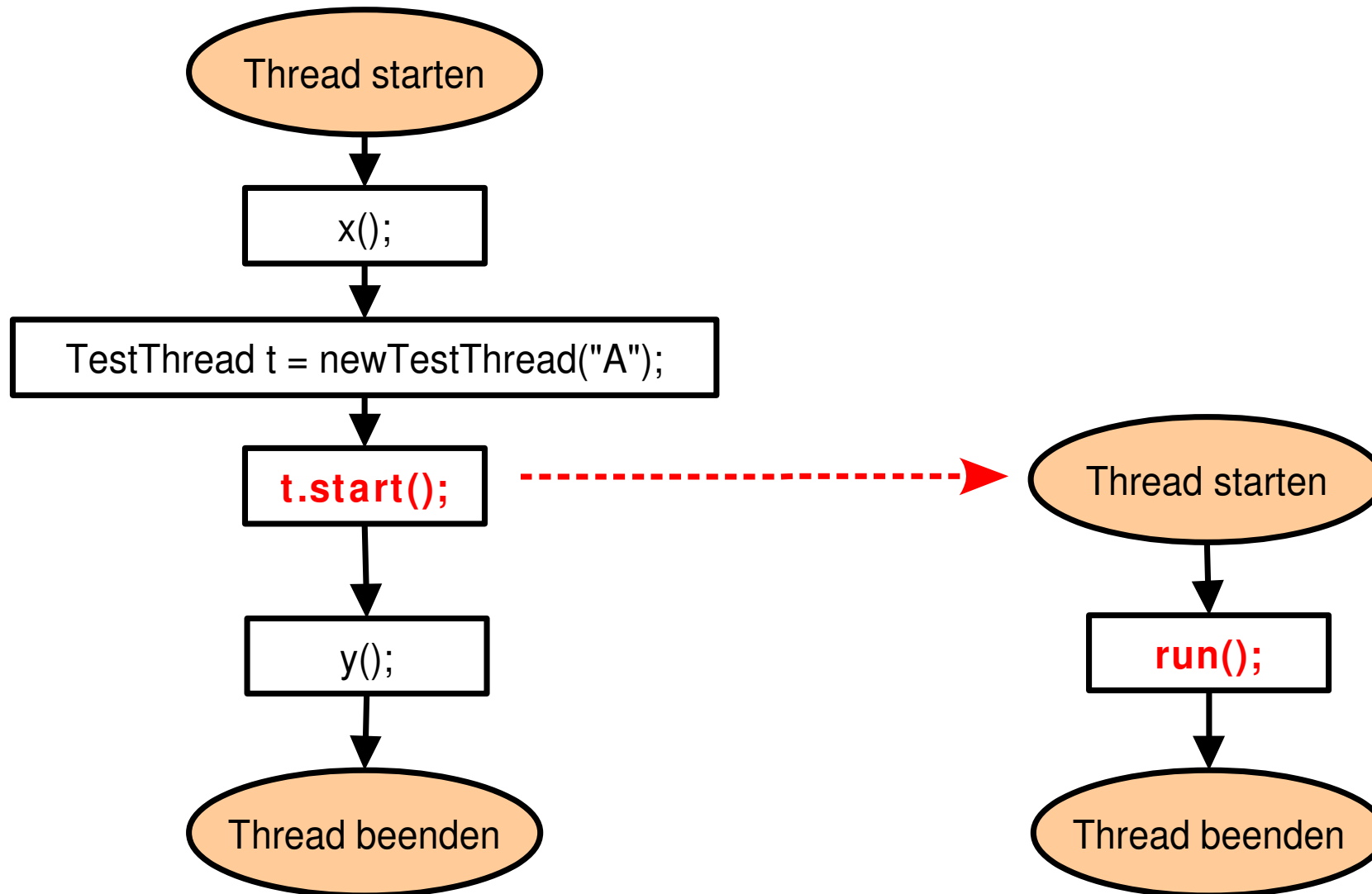


Multithreading - Beispiel

```
public class TestThread extends Thread {  
    String name;  
    public TestThread(String n) {  
        name = n;  
    }  
    public void run() {  
        for (int j = 0; j < 10; j++)  
            System.out.print(name + j + " ");  
    }  
}
```

```
x();  
TestThread t = new TestThread("A");  
t.start();  
y();
```

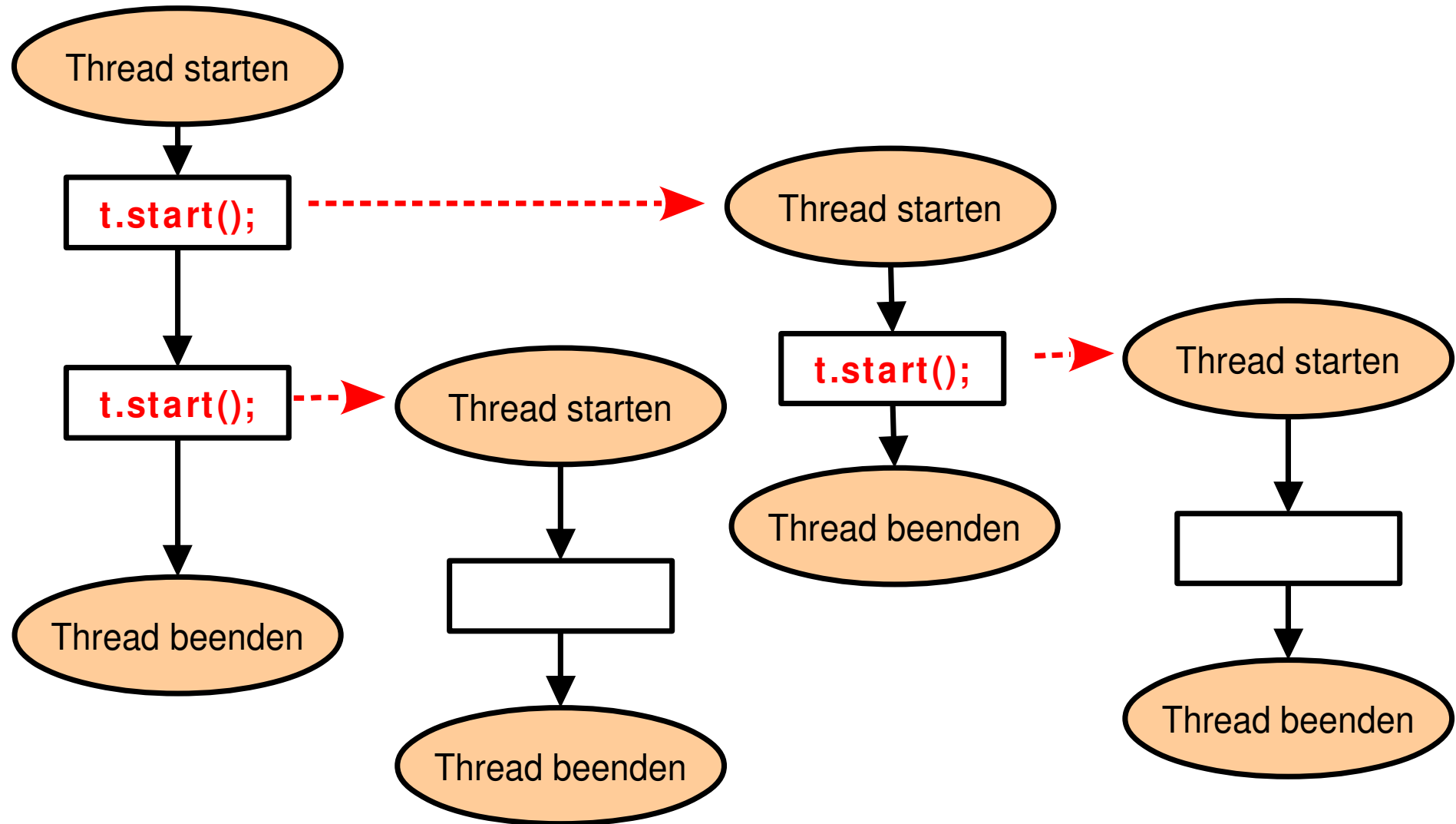
Einen weiteren Thread starten



Threads starten

- x Erzeugt man eine Instanz der Klasse Thread und ruft deren Methode start auf, so passiert folgendes:
 - ⇒ Ein neuer Thread wird gestartet.
 - ⇒ Der neue Thread führt die Methode run des Thread-Objekts aus.
 - ⇒ Der aufrufende Thread fährt gleichzeitig mit der Ausführung des eigenen Programms fort.
- x Von jedem Thread aus können neue Threads erzeugt werden.

Threads starten



Ende des Programms

- x Die Programmausführung durch die Java Virtual Machine endet, wenn alle Threads beendet wurden.
 - ⇒ genauer: nach der Beendigung aller "normalen Threads" (non-daemon)
- x Mit dem Befehl `System.exit()` kann die JVM ebenfalls beendet werden.

Erläuterungen zu Threads

- x In einer Java Virtual Machine laufen im allgemeinen mehrere Threads gleichzeitig.
- x Verschiedene Threads können auf gemeinsame Daten zugreifen. Sie können
 - ⇒ auf die gemeinsame Objekte zugreifen
 - ⇒ die gleichen Methoden aufrufen
 - ⇒ die gleichen Klassenattribute verwenden
- x Lokale Variablen sind Thread-spezifisch organisiert. Jeder Thread verfügt über einen eigenen Stack zur Speicherung seiner lokalen Variablen.

... Erläuterungen zu Threads

- x Es kann nichts darüber ausgesagt werden, mit welcher Geschwindigkeit welcher Thread ausgeführt wird.
 - ⇒ Die JVM versucht die Threads gleichmäßig auszuführen.
 - ⇒ Mit welcher Geschwindigkeit welcher Thread ausgeführt wird, kann jedoch nicht vorhergesagt werden.
 - ⇒ Unterschiedliche Programmdurchläufe können deshalb zu unterschiedlichen Ergebnissen führen!!

1.2 Synchronisation mit synchronized


Reentrant Programcode

- x Programcode wird in der Informatik allgemein als reentrant bezeichnet, wenn er von mehreren Prozessen/Threads gleichzeitig ausgeführt werden kann.
- x In der Programmiersprache Java ist Programcode von Grund auf reentrant.
 - ⇒ Genauere Betrachtung:
Von Grund auf sind alle Methoden reentrant, es ist jedoch möglich, die gleichzeitige Ausführung einer Methode durch mehrere Threads explizit zu unterbinden.


Beispiel

```
public class Counter {  
    public int c = 0;  
    public void inc() {  
        c = c + 1;  
        System.out.println(c);  
    }  
}
```


Beispieldurchlauf 1



```
c = c + 1;  
System.out.println(c);
```



```
c = c + 1;  
System.out.println(c);
```



```
c = c + 1;  
System.out.println(c);
```

 - - - - - Ausgabe: 1


```
c = c + 1;  
System.out.println(c);
```




```
c = c + 1;  
System.out.println(c);
```

 - - - - - Ausgabe: 2

Beispieldurchlauf 2



```
c = c + 1;  
System.out.println(c);
```



```
c = c + 1;  
System.out.println(c);
```



```
c = c + 1;  
System.out.println(c);
```



```
c = c + 1;  
System.out.println(c);
```

 - - - - - Ausgabe: 2

```
c = c + 1;  
System.out.println(c);
```

 - - - - - Ausgabe: 2

Beispiel

```
public class Counter {  
    private int c = 0;  
  
    public synchronized void inc() {  
        c = c + 1;  
        System.out.println(c);  
    }  
}
```

synchronized

Wird bei Objektmethoden einer Klasse *synchronized* angegeben, so wird von der JVM für jede Instanz der Klasse sichergestellt, dass zu jedem Zeitpunkt immer nur maximal eine seiner *synchronized*-Methoden von einem Thread ausgeführt wird.

Erläuterungen:

- x Erreicht ein Thread die Stelle, an der eine *synchronized*-Methode aufgerufen werden soll, so muss zunächst geprüft werden, ob gerade schon ein anderer Thread eine *synchronized* Methode des gleichen Objekts ausführt.
- x Wenn dem, so ist, so muss der Thread in einer Warteschlange des Objekts warten.
- x Er wird so lange blockiert, bis alle Threads vor ihm in der Warteschlange ihre *synchronized*-Methoden aufgerufen haben.
- x Erst dann wird die Blockierung aufgehoben, und der Thread kann mit der Ausführung der Methode beginnen.

Anmerkungen

- x Es ist sichergestellt, dass mehrere synchronized-Methoden eines Objekts nicht gleichzeitig ausgeführt werden können.
- x Das gleichzeitige Ausführen von synchronized-Methoden verschiedener Objekte ist hingegen möglich und zwar auch dann, wenn es sich um Instanzen der gleichen Klasse handelt.
- x Sind alle Methoden eines Objekts synchronized, so ist sichergestellt dass innerhalb der Ausführung einer dieser Methoden außer dem aufrufenden Prozess kein anderer Thread auf die private-Attribute des Objekts zugreifen kann.
- x Vor der Ausführung einer synchronized-Methode kann es passieren, dass ein Thread vorübergehend blockiert wird. Solange ein Thread blockiert wird, verbraucht er keine Rechenleistung (Prozessorzeit).

Alternative: synchronized-Block

```
Counter x = new Counter();  
synchronized (x) {  
    x.c = x.c + 1;  
    System.out.println(x.c);  
}
```

- x Durch einen synchronized-Block kann man den gleichen Effekt erreichen wie durch eine synchronized-Methode.
- x Der synchronized-Block bezieht sich auf
 - ⇒ ein Objekt, das explizit angegeben wird
 - ⇒ ein Stück Programmcode
- x Der Programmcode im Rumpf des synchronized-Blockes wird behandelt wie eine synchronized-Methode des Objekts.

Der Monitor des Objekts

- x Man spricht davon, dass ein Thread den Monitor eines Objekts besitzt, wenn es
 - ⇒ gerade eine synchronized-Methode des Objekts ausführt oder
 - ⇒ sich in einem synchronized-Block eines Objekts befindet.
- x Zur gleichen Zeit besitzt immer nur maximal ein Thread den Monitor eines Objekts.

1.3 Synchronisation mit wait-notify

wait, notify, notifyall

- x wait und notify sind Methoden der Klasse Object.
- x Ein Thread, der den Monitor eines Objekts besitzt, kann sich durch Aufruf der Methode wait() vorübergehend in eine Warte-Position versetzen.
- x Durch den Aufruf der wait-Methode gibt der Thread den Monitor vorübergehend ab.
 - ⇒ Folge: Ein anderer Thread kann einen synchronized-Abschnitt betreten.
- x Ein Thread, der den Monitor eines Objekts besitzt, kann die Methode notify() aufrufen. Dadurch wird der an diesem Objekt am längsten wartende Thread freigegeben und kann fortgesetzt werden sobald der Monitor des Objekts wieder abgegeben wurde.
- x notifyAll funktioniert wie notify nur dass alle wartenden Thread freigegeben werden.

Beispiel: Nachrichtenverteiler

```
public class Nachrichtenverteiler {  
    private String nachricht;  
  
    public synchronized void senden(String s) {  
        nachricht = s;  
        notifyAll();  
    }  
  
    public synchronized String empfangen() {  
        wait();  
        return nachricht;  
    }  
}
```

Erläuterungen

- x Ein Nachrichtenverteiler-Objekt wird dazu verwendet, String-Daten von einem Thread an eine Menge von Threads weiterzuleiten.
 - ⇒ empfangen bedeutet: die nächste Meldung entgegennehmen.
 - ⇒ senden bedeutet: eine Nachricht an alle schicken, die gerade auf eine Nachricht warten.
- x Rufen ein oder mehrere Threads nacheinander die Methode empfangen auf, so werden sie zunächst in den Wartezustand versetzt.
- x Sobald ein Thread die Methode senden aufruft, so geschieht folgendes:
 - ⇒ Der Thread schreibt seine Nachricht in das Attribut *nachricht* des Objekts.
 - ⇒ Die auf eine Nachricht wartenden Threads werden freigegeben.
 - ⇒ Die freigegebenen Threads geben den Wert des Attribut *nachricht* zurück.

1.4 Anwendungsszenarien

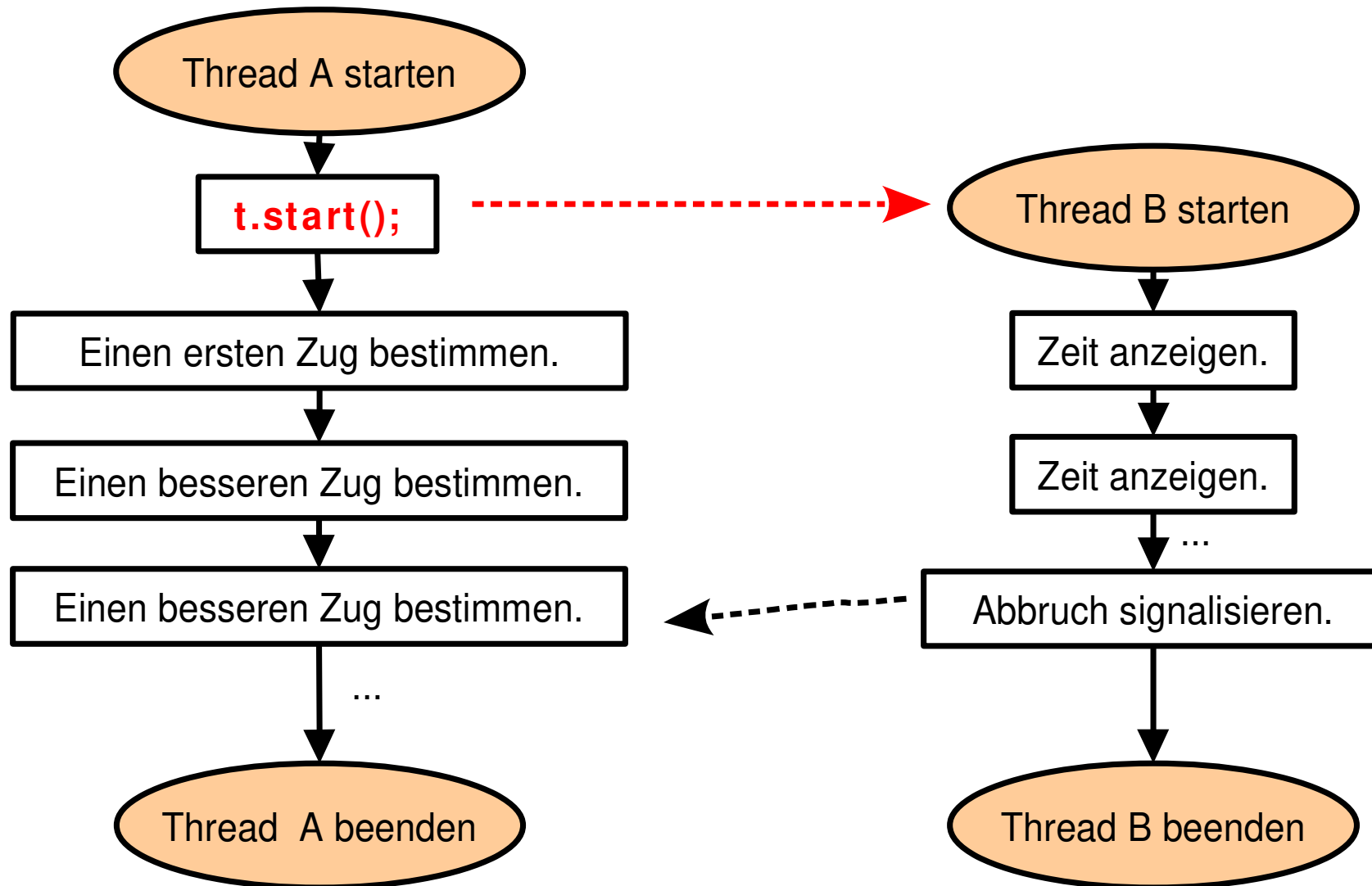
Szenario 1: Aufgabenstellung, in der nebenläufige Vorgänge beschrieben werden

- x Oft werden bereits in der Spezifikation nebenläufige Vorgänge beschrieben.
- x Folge: Es bietet sich an, die nebenläufigen Vorgänge durch je einen Thread zu realisieren.

Beispiel: Schachprogramm

- x Ein Programm soll den bestmöglichen Zug berechnen.
- x Gleichzeitig soll auf dem Bildschirm die bereits verstrichene Denkzeit des Programms angezeigt werden.
- x Nach spätestens 5 min soll ein Zug vom Schachprogramm ausgeführt werden.
- x Lösungsansatz:
 - ⇒ Das Schachprogramm (Thread A) startet einen zweiten Thread (Thread B)
 - ⇒ Thread B stellt die verstrichene Zeit auf dem Bildschirm dar während Thread A nach dem besten Zug sucht.
 - ⇒ Sobald 5 min verstrichen sind, signalisiert dies Thread B an Thread A.

Beispiel: Schachprogramm



Szenario 2: Beschleunigung durch "Parallelisierung"

- x Durch die echt gleichzeitige Ausführung von Programmteilen können Programme beschleunigt werden.
- x Voraussetzung: Die Threads können tatsächlich gleichzeitig ausgeführt werden.
 - ⇒ Mehrere Mikroprozessoren
 - ⇒ Mehrere Hardware-Threads
 - ⇒ Dienste auf externen Computern
- x Dieser Ansatz macht keinen Sinn, wenn man nur einen einzigen, simplen Mikroprozessor ohne Hardware-Threads hat.

Vorgehensweise: Teile und Herrsche

- x Das Problem wird in mehrere Teilaufgaben aufgeteilt.
- x Jede Teilaufgabe wird von einem anderen Thread bearbeitet.
- x Die Teilergebnisse werden zusammengeführt.

Szenario 3: Nebenläufige Dienste

- x Mit Hilfe von Multi-Threading ist es möglich, Dienste (Services) zu realisieren, die von mehreren Dienstnehmern gleichzeitig verwendet werden können.
- x Beispiele:
 - ⇒ Web-Service
 - ⇒ Fileserver
 - ⇒ Datenbank
 - ⇒ ...
- x Jeder Client baut eine Verbindung mit dem Service auf. Der Service muss mit mehreren gleichzeitig kommunizieren. Für jede Verbindung verwendet der Service einen eigenen Thread.

Kapitel 2

Socket-Programmierung

Motivation

- x In dieser Vorlesung werden Programmierkonzepte vorgestellt, die auf Netzwerktechnologien aufbauen. In diesem Kapitel soll deshalb die Grundmechanismen erläutert werden, mit denen Programme über ein TCP/IP-basiertes Netzwerk miteinander kommunizieren können.
 - ⇒ Bezeichnung: Socket-Programmierung
- x Die hier vorgestellten Mechanismen bilden die Grundlage für alle weiteren netzbasierten Konzepte in Java wie Servlets und RMI.
- x Abgrenzung: Das Netzwerk wird lediglich aus Sicht des Benutzers betrachtet.

2.1 Kommunikation über TCP/IP

Computer miteinander verbinden

- x Computer können auf unterschiedliche Weise technisch miteinander verbunden werden: Ethernet (IEEE 802.3), WLAN (IEEE 802.11), DSL (PPP), EDGE, UMTS, LTE, ...
- x In jedem dieser Netzwerke legt ein Netzwerkprotokoll fest, mit welchen Signalen die Kommunikationspartner Datenverbindungen herstellen und miteinander Daten austauschen können.
- x Das Internet besteht aus einer großen Anzahl von solchen Netzwerken, die miteinander verbunden sind. Netzwerkteilnehmer (Computer), die mit mehreren Netzwerken verbunden sind, werden die Daten zwischen den Netzwerken weitergereicht. Die Teilnetzwerke werden so zu einem großen Netzwerk verbunden.
- x Heute sind die meisten Computer direkt oder indirekt in einem großen, heterogenen Netz miteinander verbunden.

... Computer miteinander verbinden

- x Die Kommunikation erfolgt im Internet über ein standardisiertes Protokoll mit dem Namen TCP/IP. TCP/IP ist eine Schicht, die auf den Netzwerkprotokollen der Teilnetze aufsetzt.
- x Der TCP/IP-Stack ist eine einheitliche Programmierschnittstelle (API), die auf allen teilnehmenden Computern installiert sein muss.
- x Über diese Programmierschnittstelle können die Programme auf den teilnehmenden Computern über die Teilnetze hinweg auf einheitliche Weise miteinander kommunizieren: Verbindungen aufbauen, Verbindungen abbauen, Daten senden, Daten empfangen.

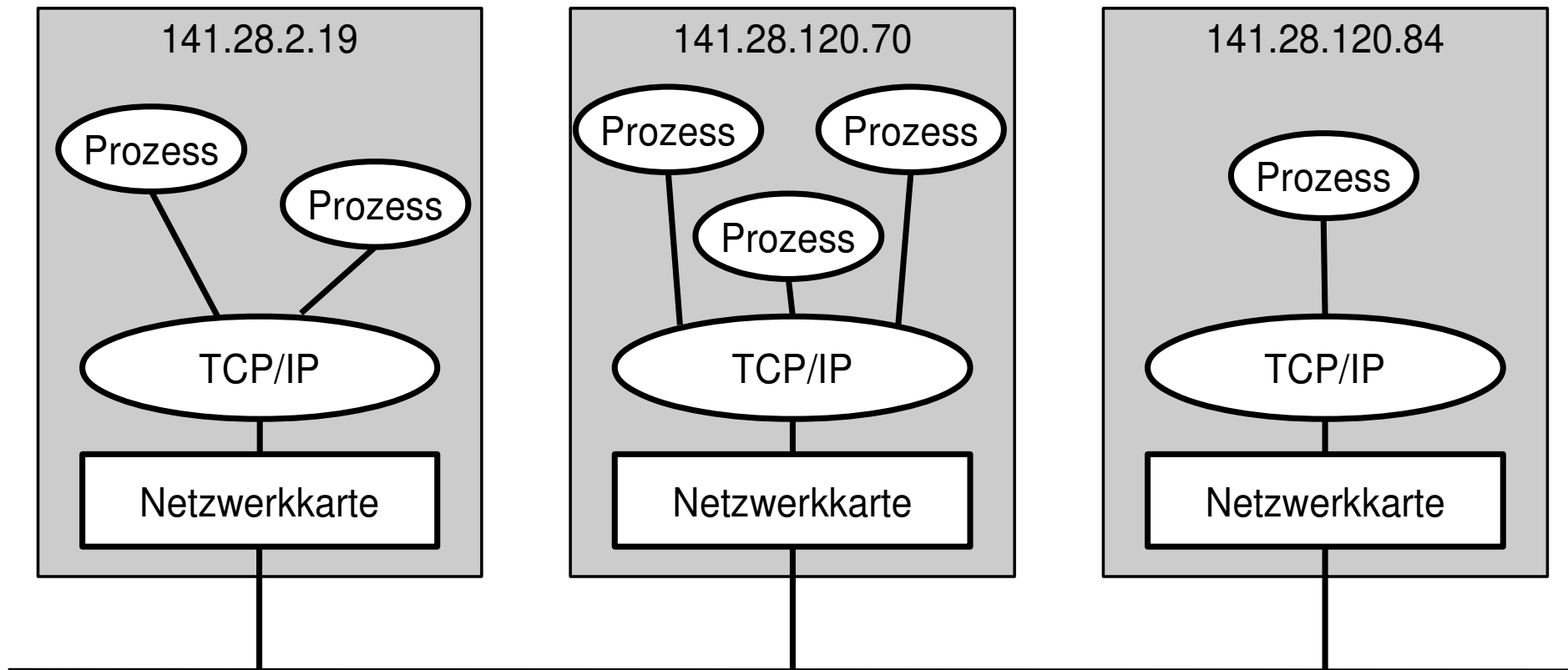
Voraussetzungen

- x Die Computer sind über ein Netzwerk direkt oder indirekt miteinander verbunden.
- x Auf jedem der Computer ist der TCP/IP-Stack installiert.
- x Jedem Computer wurde eine Internet-Adresse zugewiesen.

Daraus folgt:

- x Zwei Prozesse, die auf unterschiedlichen Computern ausgeführt werden, können über TCP/IP eine Verbindung aufbauen und anschließend über diese Verbindung miteinander kommunizieren.

Voraussetzungen



Ports, Portnummern

- x Die Teilnehmer der TCP/IP-Kommunikation sind die auf den Computern laufenden Prozesse.
- x IP-Adressen dienen der Adressierung der Computer aus Sicht des Netzwerks. Auf jedem Computer befinden sich in der Regel mehrere Prozesse. Zur eindeutigen Identifikation der Kommunikationsteilnehmer innerhalb eines Computers werden Portnummern eingesetzt.
- x Bevor ein Prozess mit einem anderen Prozess auf einem anderen Computer über TCP/IP kommunizieren kann, muss er vom TCP/IP-System eine Portnummer anfordern. Jede Portnummer wird auf jedem Computer nur einmal vergeben. Das TCP/IP-System eines jeden Computers verwaltet eine Liste mit den auf diesem Computer bereits vergebenen Portnummern. Zu jeder Portnummer wird in dieser Liste festgehalten, an welchen Prozess sie vergeben wurde.

... Ports, Portnummern

- x Als einen Port bezeichnet man die eindeutige Adresse eines Kommunikations-Teilnehmers, die sich aus IP-Adresse und Portnummer zusammensetzt.
 - ⇒ Schreibweise: *IP-Adresse:PortNummer*
- x Datenströme, die an eine Adresse der Form *IP-Adresse:PortNummer* gerichtet sind, finden über die IP-Adresse den Weg zum richtigen Computer und über die Portnummer innerhalb des Computers den Weg zum richtigen Prozess.
- x Anmerkung: Ein Prozess kann auch mehrere Ports reservieren.

TCP/IP-Verbindung, Sockets

- x Der Datenaustausch erfolgt über eine TCP/IP-Verbindung. Eine TCP/IP-Verbindung besteht immer zwischen genau zwei Teilnehmern. Die beiden Teilnehmer werden jeweils durch ihre IP-Adresse und ihren Port eindeutig identifiziert.
- x Eine TCP/IP-Verbindung ist eine Voll-Duplex-Verbindung.
 - ⇒ Jeder der beiden Teilnehmer kann zu jedem Zeitpunkt Daten senden und Daten empfangen.
- x Die beiden Endpunkte einer Verbindung bezeichnet man als Sockets. Sockets stellen Sende- und Empfangsoperationen zur Verfügung, über die der Prozess die Verbindung nutzen kann.

Verbindung, Verbindungsaufbau

Jede Kommunikation zwischen zwei Kommunikationsteilnehmern setzt sich aus drei Phasen zusammen:

- ⇒ Verbindungsaufbau
- ⇒ Datenaustausch
- ⇒ Verbindungsabbau

Initiator, Beantworter

Beim Verbindungsaufbau nehmen die beiden Teilnehmer unterschiedliche Rollen wahr, die man als Initiator und Beantworter bezeichnet ¹. Der Ablauf des Verbindungsaufbaus:

- ⇒ Der Initiator signalisiert dem Beantworter über das Netzwerk den Wunsch, mit ihm eine Verbindung aufzubauen.
- ⇒ Sobald der Beantworter anschließend signalisiert, dass er die Verbindung annehmen möchte, ist die Verbindung hergestellt.

Nach dem Verbindungsaufbau ist die Unterscheidung zwischen Initiator und Beantworter nicht mehr von Bedeutung. Beide haben dann die gleichen Rechte und Möglichkeiten und können insbesondere Daten schicken, Daten empfangen und die Verbindung beenden.

¹ Englischsprachige Begriffe: Initiator=Initiator, Beantworter=Responder
Analogie zum Telefonieren: Initiator=Anrufer, Beantworter=Angerufene

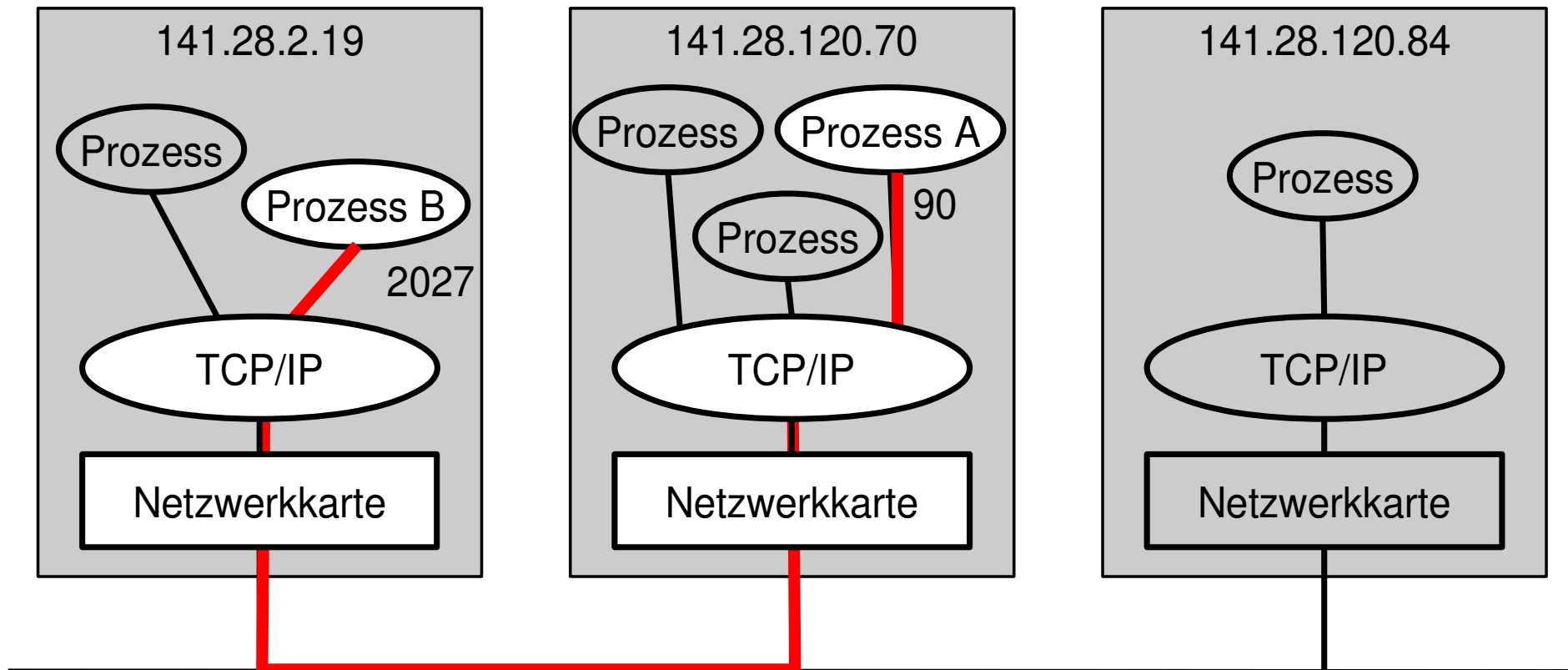
Beispiel-Szenario

- x Prozess A läuft auf dem Computer mit der IP-Adresse 141.28.120.70.
- x Prozess A reserviert sich auf seinem Computer die Port-Nummer 90.
- x Prozess A wartet darauf, dass jemand über diesen Port mit ihm eine Verbindung aufbauen möchte.
- x Prozess B läuft auf einem anderen Computer mit der IP-Adresse 141.28.2.19.
- x Prozess B möchte eine Verbindung mit 141.28.120.70:90 aufbauen. Um über TCP/IP kommunizieren zu können, benötigt auch Prozess B zunächst eine Portnummer auf seinem Computer. Er reserviert sich entweder eine ganz bestimmte Nummer oder er bekommt eine beliebige freie Nummer zugeteilt. Man nehme an, er habe die Portnummer 2027 für sich reserviert. Dann initiiert er einen Verbindungsaufbau mit 141.28.120.70:90.
- x Prozess A bekommt über das Netzwerk signalisiert, dass ein Teilnehmer mit ihm eine Verbindung aufbauen möchte. Er nimmt die Verbindung an.

... Beispiel-Szenario

- x Die Verbindung zwischen 141.28.120.70:90 und 141.28.2.19:2027, also zwischen Prozess A und Prozess B ist hergestellt.
- x Die Verbindung zwischen 141.28.120.70:90 und 141.28.2.19:2026, also zwischen Prozess A und Prozess B ist hergestellt.
- x Im Rahmen der beiden Operationen Verbindungsaufbau initiieren (Prozess B) und Verbindung annehmen (Prozess A) haben die beiden Prozesse je einen Socket erhalten - die beiden Endpunkte der Verbindung. Über diese Sockets können sie ab sofort Daten an den Kommunikationspartner verschicken, Daten von dem Kommunikationspartner empfangen und die Verbindung schließlich beenden.

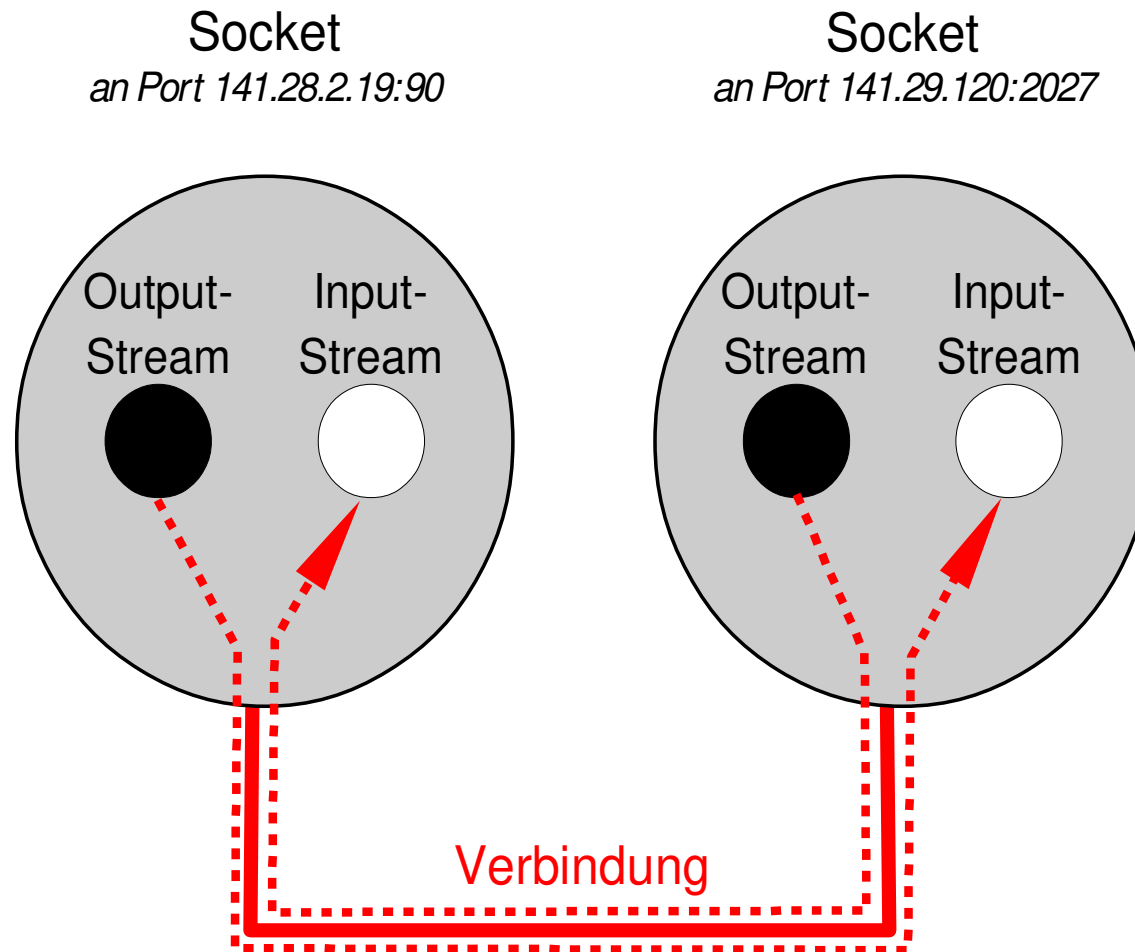
Verbindungsaufbau



Datenübertragung

- x Ist der Verbindungsaufbau erfolgreich, so erhalten der Initiator und der Beantworter jeweils einen Socket. Sockets sind die Endpunkte der Verbindung. Da TCP/IP nur Verbindungen mit genau zwei Teilnehmern kennt, sind jeder Verbindung genau zwei Sockets zugeordnet.
- x Ein Socket setzt sich immer aus einem Input-Stream und einem Output-Stream zusammen.
- x Der Output-Stream dient dem Versenden der Daten, der Input-Stream dem Empfangen von Daten.

Datenübertragung



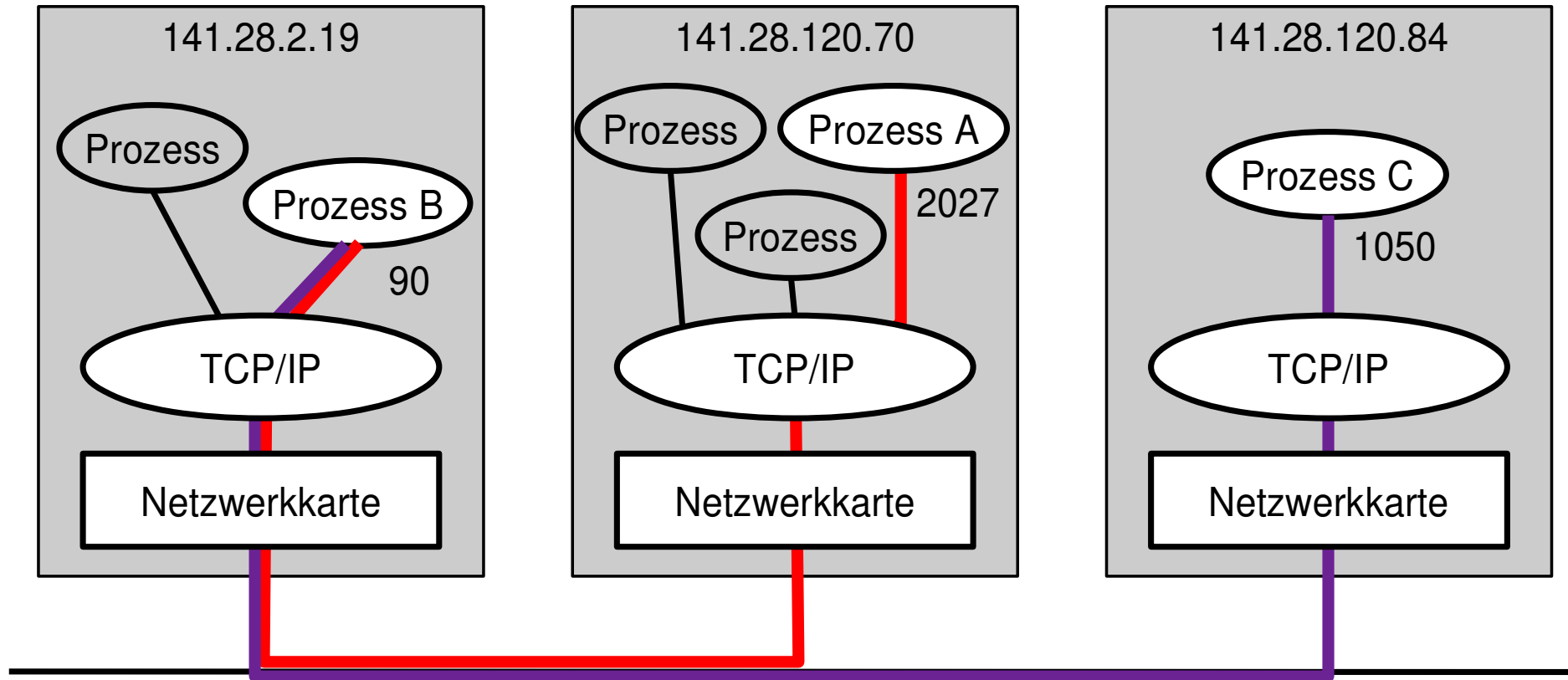
Verbindungsabbau

- x Beide Kommunikationspartner können zu jedem Zeitpunkt die Verbindung beenden.
- x Hierzu stellt der Socket eine entsprechende Operation zur Verfügung.

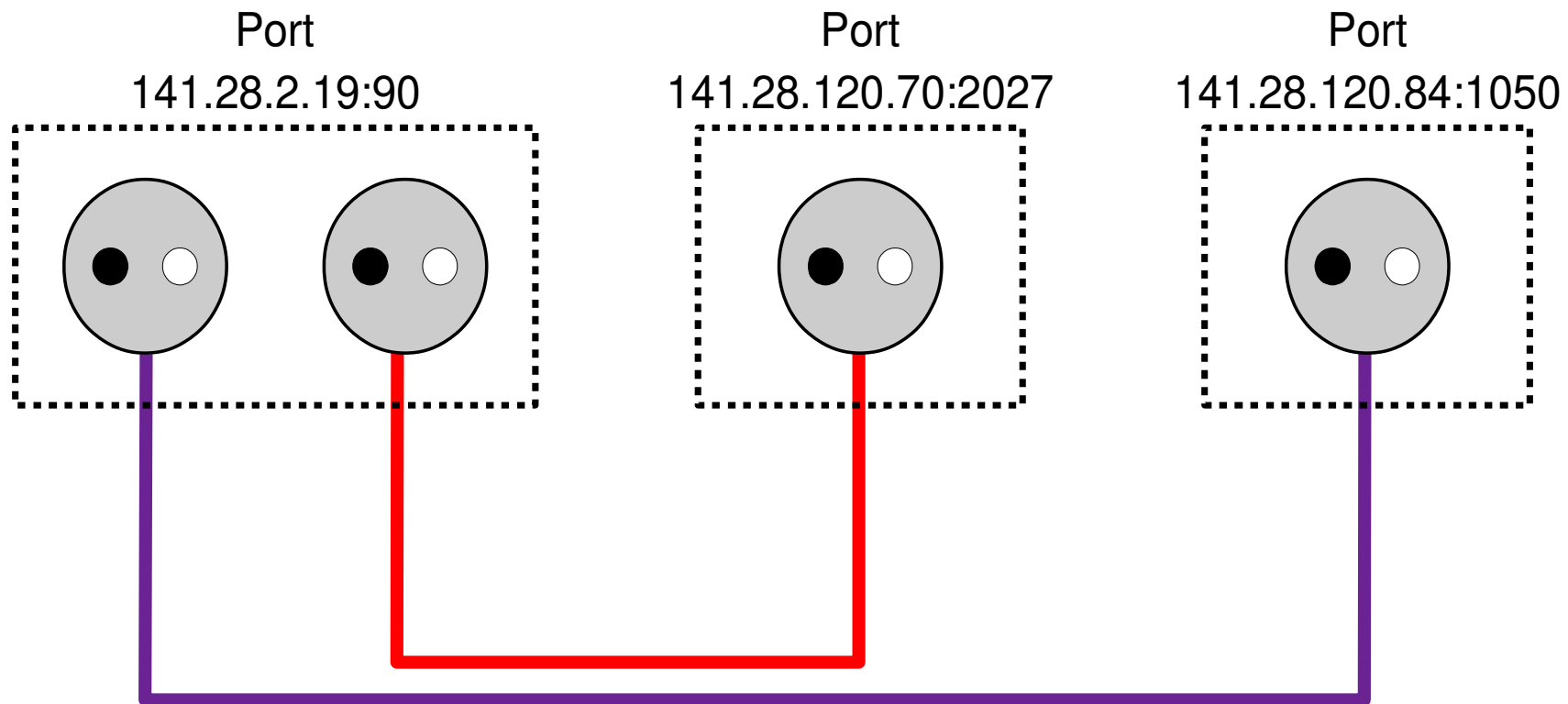
Ports, Verbindungen, Sockets

- x Zu einem Port können gleichzeitig mehrere Verbindungen aufgebaut werden.
- x Der Prozess erhält für jede Verbindung einen eigenen Socket, über die er die jeweiligen Sende- und Empfangsoperationen für diese Verbindung ausführen kann.

Mehrere Verbindungen



Mehrere Verbindungen



DNS

- x In aller Regel steht in einem TCP/IP-Netzwerk auch ein DNS-Service zur Verfügung.
- x Dann kann anstelle der IP-Adresse auch der Namen des Rechners angegeben werden. Der Name wird dann vom DNS-Service in eine IP-Adresse übersetzt.
 - ⇒ Beispiel: `www.fh-furtwangen.de` wird übersetzt in `141.28.2.19`

2.2 Sockets in Java

Socket

Instanzen der Klasse Socket realisieren die Endpunkte einer TCP/IP-Verbindung (Paket java.net).

Instanzen der Klassen InputStream und OutputStream realisieren Datenströme (Paket java.io).

- ⇒ über InputStream können Daten eingelesen werden
- ⇒ über OutputStream können Daten geschrieben werden

Jedes Socket-Objekt enthält eine Instanz von InputStream und eine Instanz von OutputStream.

- ⇒ Zugriff über die Objektmethoden `getInputStream()` und `getOutputStream()`

Offene Frage:

- ⇒ Wie baut man eine Verbindung auf?
- ⇒ Wie kommt man zu Instanzen von Socket?

Verbindungsaufbau aus Sicht des Beantworters

Erzeugen einer Instanz von `ServerSocket` mit einer Portnummer als Parameter.

Aufruf der Objektmethode `accept()`:

- ⇒ Warten auf einen Verbindungswunsch (Blockierung)
- ⇒ Sobald ein Verbindungswunsch eingeht, wird ein `Socket`-Objekt zurückgegeben.

```
ServerSocket serverSocket = new ServerSocket(90);  
Socket socket = serverSocket.accept();
```

Verbindungsaufbau aus Sicht des Initiators

Aufruf des Konstruktors von Socket mit der IP-Adresse und der Portnummer der Gegenstelle als Parameter.

```
Socket socket = new Socket("141.28.2.19", 90);
```

Anmerkungen:

- ⇒ Dem Initiator wird im Rahmen dieses Aufrufs eine beliebige freie Portnummer zugeteilt. Andere Konstruktoren der Klassen Socket erlauben es, dass man auch eine ganz bestimmte Port-Nummer reserviert. Im allgemeinen ist jedoch die Portnummer des Initiators aus Sicht des Programmes von geringer Bedeutung.
- ⇒ Anstelle der IP-Adresse darf auch der Host-Name verwendet werden.

BufferedReader, BufferedWriter

Die Instanzen von `InputStream` und `OutputStream` bieten eine elementare, ungepufferte Kommunikation an, die sich auf einzelne Bytes beschränkt.

Zur einfachen Programmierung textbasierter Anwendungen empfiehlt sich der Einsatz von `BufferedReader` bzw. `BufferedWriter`. Über diese beiden Klassen kann, aufbauend auf `InputStream` und `OutputStream`, eine zeilenweise, gepufferte Kommunikation hergestellt werden kann.

```
BufferedWriter writer =  
    new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
  
BufferedReader reader =  
    new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

Daten übertragen, Verbindung schließen

<code>writer.write(String s);</code>	Zeichenkette senden
<code>writer.newLine();</code>	Zeilenumbruch senden
<code>writer.flush();</code>	Die mit write gesendeten Daten werden nicht unbedingt sofort gesendet, sondern eventuell erst gepuffert. Mit flush() werden die Daten des Puffers übertragen und der Puffer geleert.
<code>reader.readLine();</code>	Einlesen einer Textzeile Für den Fall, dass die Gegenseite die Verbindung schließt, wird der Wert <i>null</i> zurückgegeben.
<code>socket.close();</code>	Schließen der Verbindung

Anmerkung

Zur Unterscheidung der beiden Sockets einer Verbindung in Bezug auf den Verbindungsaufbau haben sich die Begriffe Client-Socket und Server-Socket etabliert, obwohl es richtiger wäre, diese als Initiator-Socket bzw. Beantworter-Socket zu bezeichnen. Die Begriffe Client und Server beziehen sich auf das Konzept der Services, auf das im nächsten Abschnitt eingegangen werden soll. Die meisten Internet-Anwendungen folgen diesem Konzept. Es gibt jedoch auch Internet-Anwendungen, in denen es keine ausgewiesene Clients und Server gibt. In diesen Anwendungen sind die Begriffe Client-Socket und Server-Socket verwirrend.

2.3 SSL-Verbindungen

SSL-Verbindung

- x SSL steht für Secure Socket Layer
- x Eine mit einer SSL gesicherte Verbindung ist eine Alternative zu einer gewöhnlichen TCP/IP-Verbindung. Nach dem Aufbau einer SSL-Verbindung werden die Daten in gleicher Weise zwischen den beiden Kommunikationspartnern ausgetauscht wie bei einer gewöhnlichen TCP/IP-Verbindung.
- x Eine SSL-Verbindung garantiert zusätzliche Sicherheitseigenschaften.

Sicherheitseigenschaften durch SSL

Eigenschaften, die eine SSL-Verbindung im Vergleich zu einer gewöhnlichen TCP/IP-Verbindung zusätzlich garantiert:

x Vertraulichkeit

⇒ Die Daten werden verschlüsselt übertragen, sodass sie von keinem dritten gelesen werden können.

x Integrität

⇒ Es wird sichergestellt, dass die Daten auf dem Übertragungsweg nicht unbemerkt verändert werden können.

x Authentizität

⇒ Mit Hilfe eines Zertifikats wird sichergestellt, dass der Server der ist, der er vorgibt zu sein. Zusätzlich kann auch eine Zertifikat-basierte Client-Authentifizierung durchgeführt werden.

Aufbau einer SSL-Verbindung durch den Initiator

```
SSLConnectionFactory factory =  
    (SSLConnectionFactory) SSLConnectionFactory.getDefault();  
SSLSocket socket =  
    (SSLSocket) factory.createSocket( hostname, port);
```

Es entsteht eine Instanz der Klasse `SSLSocket`. `SSLSocket` ist eine Sohnklasse von `Socket`. Somit kann die Variable `socket` aus obigem Programmcode in gleicher Weise verwendet werden wie die `socket`-Variablen aus den vorangegangenen Programmbeispielen.

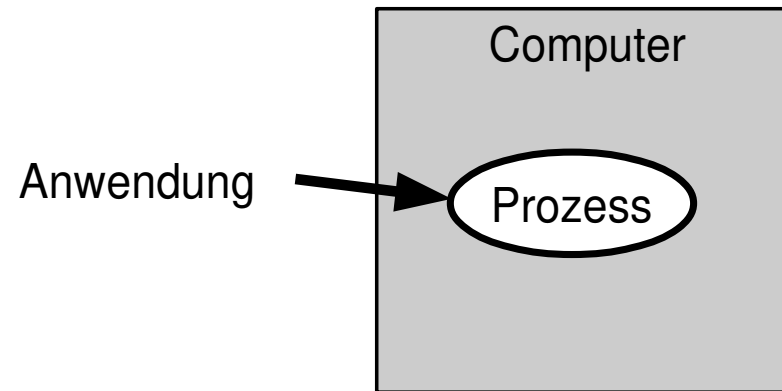
- x Mit Hilfe von *BufferedReader* und *BufferedWriter* können Daten gesendet und empfangen werden.
- x Mit *close()* kann die Verbindung getrennt werden.

Kapitel 3

Verteilte Anwendungen

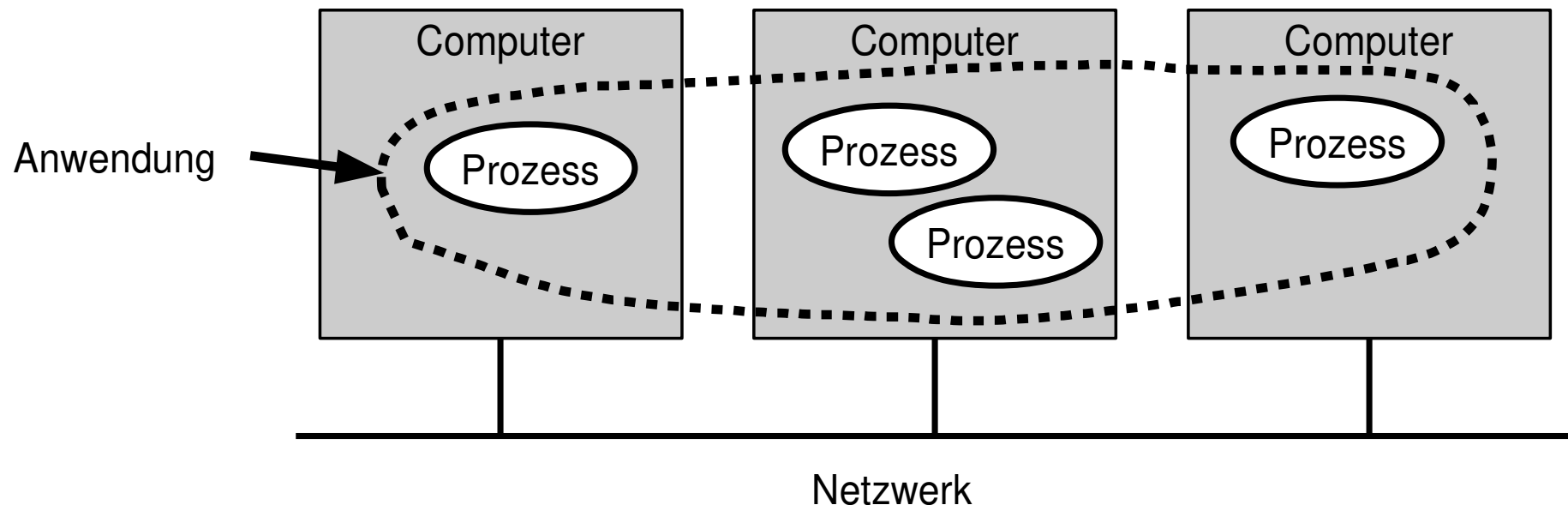
Klassische Anwendung

- x Die Anwendung besteht aus einem Programm, das auf einem Computer ausgeführt wird.



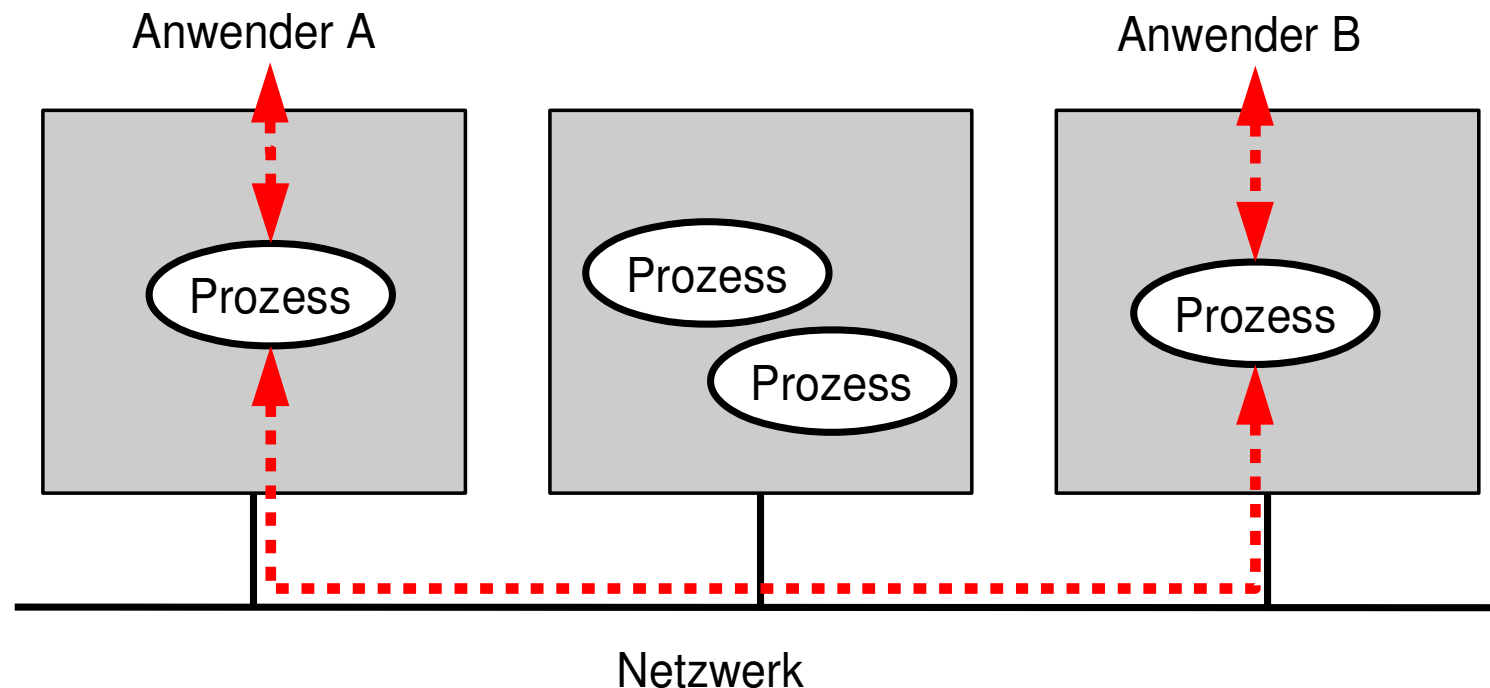
Verteilte Anwendung

- x Die Anwendung bezieht mehrere Computer ein, die über ein Netzwerk miteinander verbunden sind.
- x Auf den Computern werden Programme ausgeführt, die jeweils einen Teil der gesamten Anwendung darstellen.
- x Die Teile der Anwendung kommunizieren untereinander über das Netzwerk.



Direkte Kommunikation zwischen Anwendern

- x Anwender können in einer verteilten Anwendungen dadurch miteinander kommunizieren, dass sie jeweils auf ihrem Computer eine Anwendung starten und über diese Anwendungen miteinander Daten austauschen.



Argumente gegen eine direkte Kommunikation

- x Die Computer der Anwender müssen zum gleichen Zeitpunkt mit dem Netzwerk verbunden sein.
 - ⇒ Gerade bei mobilen Computern ist dies oft nicht gewährleistet.
- x Die Daten müssen oft nicht nur eine räumliche Distanz, sondern auch eine zeitliche Distanz überbrücken.
 - ⇒ Anwender A sendet die Daten heute, Anwender B holt sie morgen ab.
 - ⇒ Konsequenz: Zwischenspeicherung der Daten erforderlich
- x TCP/IP erlaubt lediglich die Kommunikation zwischen zwei Teilnehmern.
 - ⇒ Ein Wunsch könnte sein, dass mehrere Teilnehmer gleichzeitig miteinander kommunizieren sollen.

... Argumente gegen eine direkte Kommunikation

- x Firmendaten müssen speziell vor Verlust und unerlaubtem Zugriff geschützt werden.
 - ⇒ Ein vom Mitarbeiter umhergetragener Notebook eignet sich dazu nicht.
- x DHCP
 - ⇒ Zur Überwindung der Knappheit von IP-Adressen bei IPv4, bekommen die Computer oft dynamisch IP-Adressen zugewiesen.
 - ⇒ Wenn ein Computer keine feste Adresse hat, so eignet er sich auch nicht dazu einen Service zur Verfügung zu stellen. Clients müssen wissen, unter welcher IP-Adresse der Service zu erreichen ist.

... Argumente gegen eine direkte Kommunikation

x Masquerading

- ⇒ Zur Überwindung der Knappheit von IP-Adressen bei IPv4, werden Teilnetze gebildet, in denen Adressen vergeben werden, die nicht weltweit eindeutig sind und die nur in dem jeweiligen Teilnetz verwendbar sind.
- ⇒ Die Computer des Teilnetzes kommunizieren mit Computern des Internet indirekt über einen Gateway-Computer.
- ⇒ Vom Internet aus kann mit diesen Computern keine direkte Verbindung aufgebaut werden.

... Argumente gegen eine direkte Kommunikation

x Firewalls

- ⇒ Computer eines Firmennetzes werden gern durch Firewalls vor unberechtigtem Zugriff geschützt.
- ⇒ Beschränkung: Computer des Internet können nur bei ausgewählten Computern des Firmennetzes auf ganz bestimmte Anwender-orientierte Services wie http und https zugreifen.
- ⇒ Alle anderen Ports werden für den Zugriff von außen gesperrt. Gesperrt werden insbesondere alle Ports von Services, die der Administration dienen. Beispiel: telnet
- ⇒ Konsequenz: Computer des Firmennetzes können zwar Verbindung mit anderen Computern des eigenen Firmennetzes und des Internets initiieren - ein Computer des Internet kann jedoch in der Regel keine Verbindung mit einem Computer des Firmennetzes initiieren.

3.1 Client/Server-Konzept, Services

Das Client/Server-Konzept

Man unterscheidet bei einer Client-Server-Architektur strikt zwischen zwei Arten von Computern:

- x Client-Computer

- ⇒ sind die Arbeitsplätze für die Anwender
- ⇒ werden während der Arbeitszeit mit dem Netzwerk verbunden

- x Server

- ⇒ sind permanent mit dem Netzwerk verbunden
- ⇒ erbringen permanent zentrale Dienste
- ⇒ unterstützen mit ihren zentralen Diensten eine indirekte Kommunikation zwischen den Anwendern
- ⇒ halten die für die Organisation relevanten Daten persistent
- ⇒ sind robust und ausfallsicher konstruiert

Client/Server-Anwendung

Eine Client/Server-Anwendung besteht aus zwei Programmteilen:

- x ein Service-Programm, das permanent auf dem Server ausgeführt wird
- x einem Client-Programm, das auf allen Client-Computern ausgeführt wird, wann immer der Service vom Client-Computer aus genutzt werden soll

Services

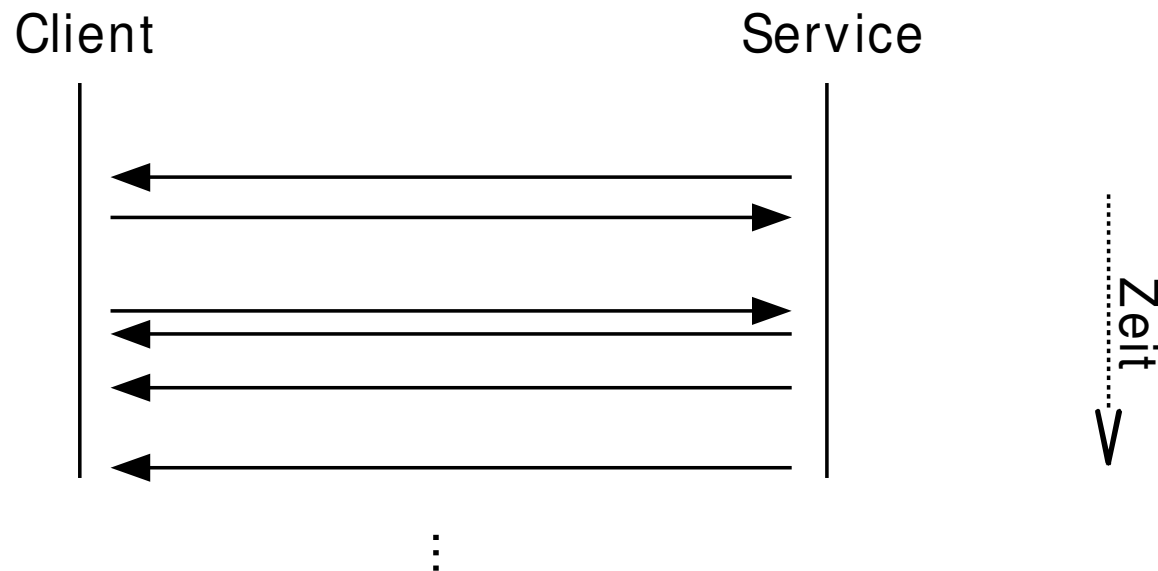
- x Ein Server soll bestimmte Services zur Verfügung stellen, die von Client-Computern aus über das Internet genutzt werden können. Dazu wird auf dem Server ein Service-Programm gestartet, das dort unbeschränkt lange läuft und den Service permanent zur Verfügung stellt.
- x Es wird ein Protokoll definiert, das festlegt, wie durch die Kommunikation über eine TCP/IP-Verbindung auf die Dienste zugegriffen werden kann.
Bezeichnung: Anwendungsprotokoll. Das Anwendungsprotokoll legt fest,
 - ⇒ welche Anfragen an den Server geschickt werden können
 - ⇒ wie der Server darauf antwortet
 - ⇒ wie die Anfragen und die Antworten in Form von digitalen Daten kodiert werden

... Services

- x Das Service-Programm verhält sich wie folgt:
 - ⇒ Es reserviert sich auf dem Server eine Portnummer.
 - ⇒ Clients bauen Verbindungen mit dem Port auf. Das Service-Programm nimmt die Verbindungen alle an.
 - ⇒ Auf jeder solchen Verbindung
 - x empfängt das Service-Programm die eingehenden Daten und interpretiert sie gemäß dem definierten Anwendungsprotokoll,
 - x bearbeitet die eingehenden Daten und
 - x schickt von Zeit zu Zeit Daten in dem durch das Anwendungsprotokoll festgelegten Format über die Verbindung zurück an den Client

... Services

- x Die Abfolge der Empfangs- und Sende-Operationen kann prinzipiell beliebig sein.
 - ⇒ Der Service kann zu beliebigen Zeitpunkten Daten verschicken.
 - ⇒ Ob und wie die Sende-Operationen des Service im Zusammenhang mit den von ihm empfangenen Daten steht, hängt vom Anwendungsprotokoll ab.

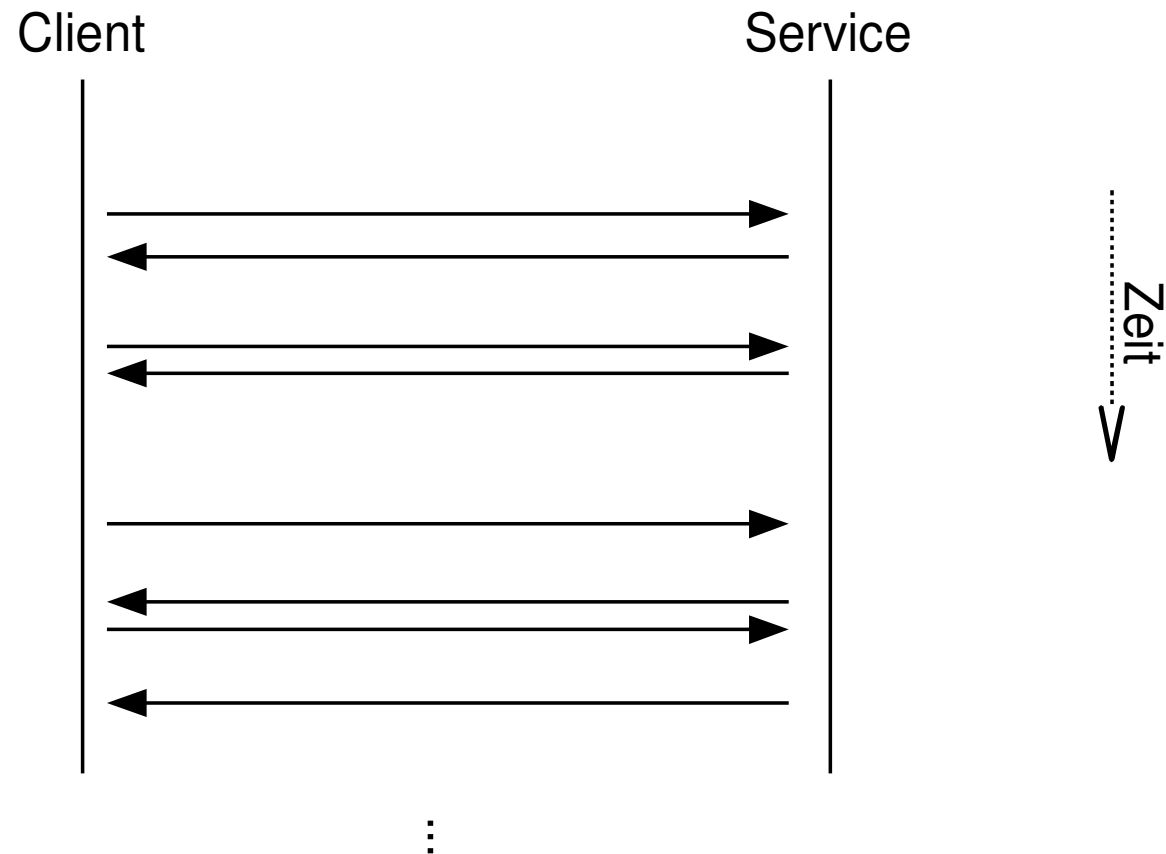


Request-Response-Prinzip

Viele Anwendungsprotokolle folgen einem ganz bestimmten Schema für den Kommunikationsablauf zwischen Client und Server, das als Request-Response-Prinzip bezeichnet wird. Das Request-Response-Prinzip:

- x Die Daten, die vom Client zum Service geschickt haben, werden als Requests bezeichnet, die Daten, die vom Service zum Client geschickt werden, werden als Responses bezeichnet. Request und Response sind Datenpakete, deren Format im Anwendungsprotokoll genau definiert ist.
- x Die zeitliche Abfolge der Kommunikation ist streng vorgegeben: Der Service antwortet auf jeden Request unmittelbar mit genau einer Response.
 - ⇒ Der nächste Request wird erst empfangen, wenn die vorige Request bereits mit einer Response beantwortet wurde.
 - ⇒ Der Service verschickt dann und nur dann Daten an den Client, wenn er von ihm einen Request erhält.

Request-Response-Prinzip



Standard-Services

- x Es gibt zahlreiche standardisierte Anwendungsprotokolle für Services.
- x Für Standard-Services wurden Default-Portnummern definiert.
- x In der Regel stellt ein Server Standard-Services an diesen Portnummern zur Verfügung. Client-Programme für diese Services fordern dann zum Verbindungsaufbau lediglich die IP-Adresse oder den Hostnamen und gehen implizit davon aus, dass die Portnummer des Service die Default-Portnummer für das Anwendungsprotokoll ist, das sie zur Kommunikation mit dem Service verwenden.

Beispiele für Standard-Services

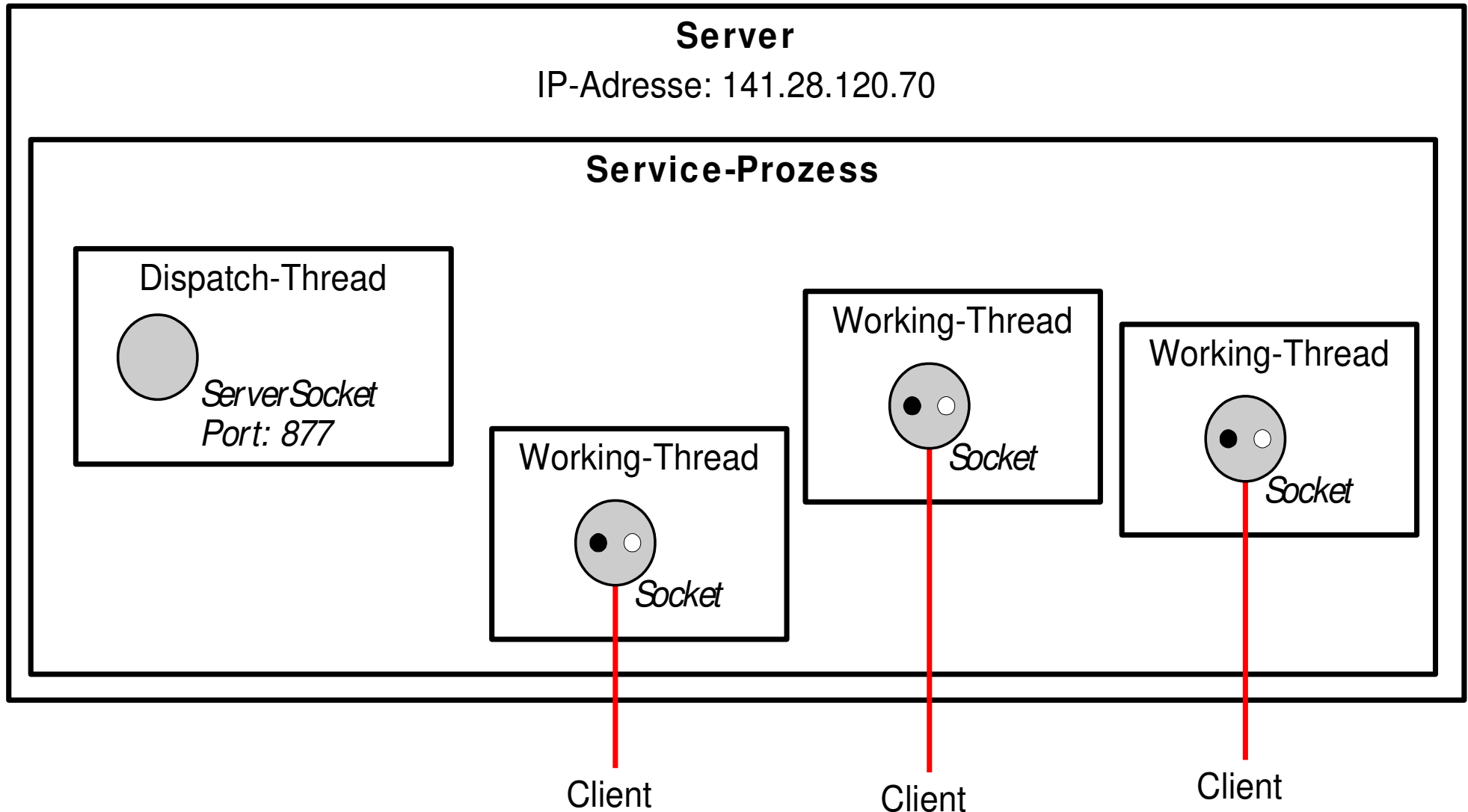
<i>Service</i>	<i>Bedeutung</i>	<i>Default-Portnummer</i>
http	hypertext transfer protocol	80
telnet	Terminal emulator	23
smtp	simple mail transfer protocol	25
ftp	file transfer protocol	20,21
https	http auf der Basis von ssl	443

Services und Multithreading

In der Regel wird von einem Service gefordert, dass ihn mehrere Clients gleichzeitig verwenden können. Ein Service-Programm, das mehrere Verbindungen zu Clients gleichzeitig bedient, lässt sich am einfachsten mit Hilfe von Multithreading realisieren.

- ⇒ Das Service-Programm besteht aus einem Prozess mit mehreren Threads:
 - x ein Dispatch-Thread
 - x je ein Working-Thread pro Verbindung
- ⇒ Wenn sich der Client mit dem Service verbindet, so nimmt diese Verbindung zunächst der Dispatch-Thread entgegen. In Java geschieht dies mit einem `ServerSocket`-Objekt und mit jeder eingehenden Verbindung entsteht in Java ein `Socket`-Objekt.
- ⇒ Unmittelbar nach dem Verbindungsaufbau reicht der Dispatcher die Verbindung (in Java: `Socket`-Objekt) an einen Working-Thread weiter. Der Working-Thread ist während der gesamten Lebensdauer der Verbindung dafür zuständig, die eingehenden Daten gemäß Anwendungsprotokoll zu bearbeiten.

Service mit Multithreading



Einfache Lösung für Service mit Multithreading

- x Der Dispatch-Thread erzeugt für jede eingehende Verbindung einen neuen Working-Thread und startet ihn.
- x Der Working-Thread kommuniziert mit dem Client und beendet sich automatisch nach dem Ende der Verbindung.

Nachteil:

- x Mit jeder eingehenden Verbindung muss ein neuer Thread erzeugt werden

Thread-Pools

- x Grundlegender Gedanke: ein einmal erzeugter und gestarteter Thread soll nacheinander für mehrere Verbindungen eingesetzt werde.
- x Der Service verwaltet alle gestarteten Threads in einem Pool.
 - ⇒ Thread-Pool = Sammelbehälter für Working-Threads
- x Schon bevor die erste Verbindung eingeht, startet der Service vorsorglich eine bestimmte Anzahl von Working-Threads und tut sie in den Thread-Pool.
- x Threads, die gerade keine Verbindung bearbeiten, werden blockiert (wait) und warten innerhalb des Thread-Pools in einer Warteschlange auf den nächsten Auftrag.

... Thread-Pools

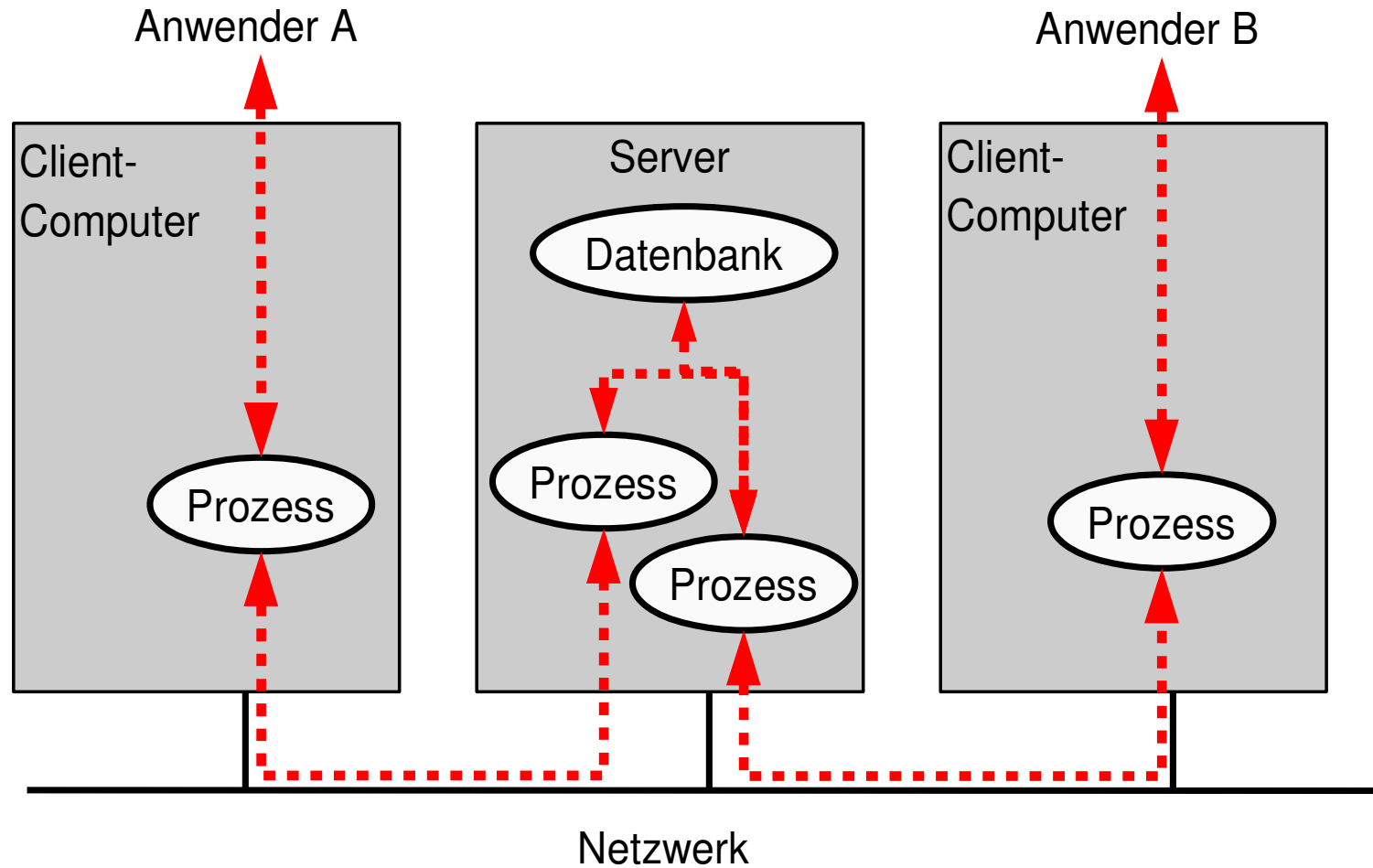
- x Sobald eine neue Verbindung entgegen genommen wird, wird der Socket dem nächsten Thread in der Warteschlange zugeordnet. Der Thread wird deblockiert (notify) und beginnt dann sofort mit dem nächsten Auftrag: der Kommunikation mit dem Client.

Indirekte Kommunikation zwischen Anwendern

Konzept:

- x Anwender bauen nicht untereinander eine Verbindung auf, sondern verbinden sich jeweils mit einem zentralen Dienst. Der angebotene Dienst unterstützt eine Übermittlung von Daten zwischen den Clients.
- x Der zentrale Service wartet ständig auf eingehende Verbindungen und nimmt diese entgegen.
 - ⇒ Rolle beim Verbindungsaufbau: Beantworter.
- x Die IP-Adresse und der Port des Dienstes sind fest und werden den Client-Anwendungen durch geeignete Konfiguration bekanntgemacht.
- x Will ein Client den Dienst benutzen, so baut er eine Verbindung mit dem Dienst auf.
 - ⇒ Rolle beim Verbindungsaufbau: Initiator

... Indirekte Kommunikation zwischen Anwendern



Konsequenzen

- x Die Anzahl der Anwender, die miteinander Daten austauschen, ist variabel und nicht auf zwei beschränkt.
- x Die Anwender müssen zum Datenaustausch nicht gleichzeitig aktiv sein.
 - ⇒ Die von einem Anwender zum Server übermittelten Daten können auch zunächst im Hauptspeicher des Servers gepuffert oder persistent abgespeichert werden - z. B. in einer Datenbank. Zu einem späteren Zeitpunkt können dann ein oder mehrere Anwender diese Daten lesen.
- x Die Daten können in einem persistenten Speicher auf dem Server dauerhaft gespeichert und protokolliert werden.

Variante einer indirekten Kommunikation zwischen Anwendern

Eine Kommunikation zwischen Anwendern kann auch dadurch hergestellt werden, dass sich die Anwender mit unterschiedlichen Servern verbinden, die dann untereinander - direkt oder indirekt - die Daten weiterleiten.

Beispiel: E-Mail.

- x Ein Anwender A verschickt eine E-Mail indem er sie von seinem Arbeitsplatzrechner an einen Server A weiterleitet. (typischerweise per smtp-Protokoll)
- x Server A leitet die E-Mail an Server B weiter. Die Adresse von Server B steht hinter dem @ in der E-Mail-Adresse.
- x Server B empfängt die E-Mail. Da der Empfänger die E-Mail nicht genau in diesem Augenblick abholt, wird sie auf Server B zunächst abgespeichert.
- x Der Adressat, ein Anwender B, verbindet sich mit seinem E-Mail-Server Server B und holt die zuletzt eingegangenen E-Mails ab. (typischerweise per pop3-Protokoll)

Optionen bei der indirekten Kommunikation zwischen Anwendern

Der Server kann die Kommunikation durch geeignete, an die geschäftlichen Erfordernisse angepasste Dienste strukturieren.

- ⇒ Definition von Transaktionen
- ⇒ Konsistenzprüfung der an den Server übermittelten Daten
- ⇒ Verwaltung von Anwendern, Zugangskontrolle über Authentifizierung
- ⇒ Definition von Rollen und rollenspezifischen Zugriffsrechten
- ⇒ Vergabe von Zugriffsrechten für bestimmte Transaktionen

3.2 Realisierung von Client/Server-Anwendungen

Client/Server-Anwendungen

- x Folgt man dem Client/Server-Konzept, so ist bei der Konstruktion einer verteilten Anwendung in einem ersten Schritt zu klären
 - ⇒ welche Teile der Anwendung auf dem Server laufen sollen (Service-Programm)
 - ⇒ welche Teile der Anwendung auf den Client-Computern laufen sollen (Client-Programm)
 - ⇒ wie das Anwendungsprotokoll aussieht, mit dem die Clients mit dem Server kommunizieren
- x Es gibt verschiedene Anwendungsprotokolle und dafür jeweils unterschiedliche Software-Systeme, auf deren Basis man eine Client-Server-Anwendung realisieren kann.

Architekturen für Client-Server-Anwendungen

- x Gegeben: die funktionale Beschreibung einer verteilten Anwendung.
 - ⇒ Use Cases – Wer soll mit dem System was tun können.

- x Schritt 1: Festlegen einer Software-Architektur
 - ⇒ Auswahl von Anwendungsprotokollen, Software-Frameworks (Application Server), Entwicklungsumgebungen

- x Schritt 2: Implementierung des anwendungsspezifischen Programmcodes
 - ⇒ Datenstrukturen (ER-Modelle), Geschäftslogik, Klassendiagramme, Programmierung

Kriterien zur Beurteilung von Architekturen für Client-Server-Anwendungen

- x Welche Einschränkungen haben die Architekturen?
 - ⇒ Interaktive Benutzeroberfläche möglich? Push-Operationen? Mobile Endgeräte? ...
- x Wie hoch ist der Entwicklungsaufwand?
 - ⇒ Eigenschaften der Application-Server
 - ⇒ Unterstützung durch Entwicklungswerkzeuge
- x Wie hoch ist die Kommunikationslast zwischen Client und Server?
- x Wie hoch ist der Ressourcen- und Rechenzeit-Bedarf auf den Client-Computer?
- x Wie hoch ist die Last auf dem Server?

Roll-Out bei verteilten Anwendungen

- x Um eine verteilte Client/Server-Anwendung in Betrieb zu nehmen,
 - ⇒ muss das Service-Programm auf einem Server installiert werden, den alle beteiligten Arbeitsplatzrechner über das Netzwerk erreichen können
 - ⇒ muss auf allen Arbeitsplatzrechnern die Client-Software installiert werden
 - ⇒ müssen der Client-Software durch geeignete Konfiguration die IP-Adresse und der Port des Service bekannt gemacht werden
- x Jede neue Version der verteilten Anwendung bedeutet im allgemeinen, dass sowohl auf dem Server als auch auf allen Arbeitsplatzrechnern das Client-Programm ausgetauscht werden muss.
 - ⇒ Der Update aller Clients ist aufwändig.
 - ⇒ Es besteht die Gefahr, dass Client-Computer mit veralteten Versionen der Client-Software auf den Server zugreifen.

Standardisierte Client-Software

- x Für viele Problemstellungen empfiehlt sich eine vereinfachte Vorgehensweise bei der Konstruktion einer verteilten Anwendung:
 - ⇒ Man verwendet eine standardisierte Client-Software, die sich für verschiedene Server-Anwendungen eignet.
 - ⇒ Die Client-Software kommuniziert über ein standardisiertes Protokoll mit einem Server, der frei konfiguriert werden kann.
 - ⇒ Man beschränkt sich darauf, die serverseitige Software zu entwickeln.
- x Beispiel: Browser für http/html
 - ⇒ Abfrage von html-Seiten mittels http
 - ⇒ Visualisierung der html-Seiten

Standardisierte Client-Software

Vorteile:

- x Die Client-Software muss nur einmal installiert werden und kann dann mit verschiedenen Server-Anwendungen kommunizieren.
- x Das Anwendungsprotokoll, über das Client und Server kommunizieren, ist standardisiert und gut dokumentiert.
- x Für die Konstruktion der serverseitigen Software stehen im allgemeinen vorgefertigte Frameworks zur Verfügung (Application Server, PHP-Plugin).

Nachteil:

- x Geringere Flexibilität bei der Konstruktion der verteilten Anwendung durch die Festlegung auf ein bestimmtes Anwendungsprotokoll für die Kommunikation.

3.3 Backend-Services

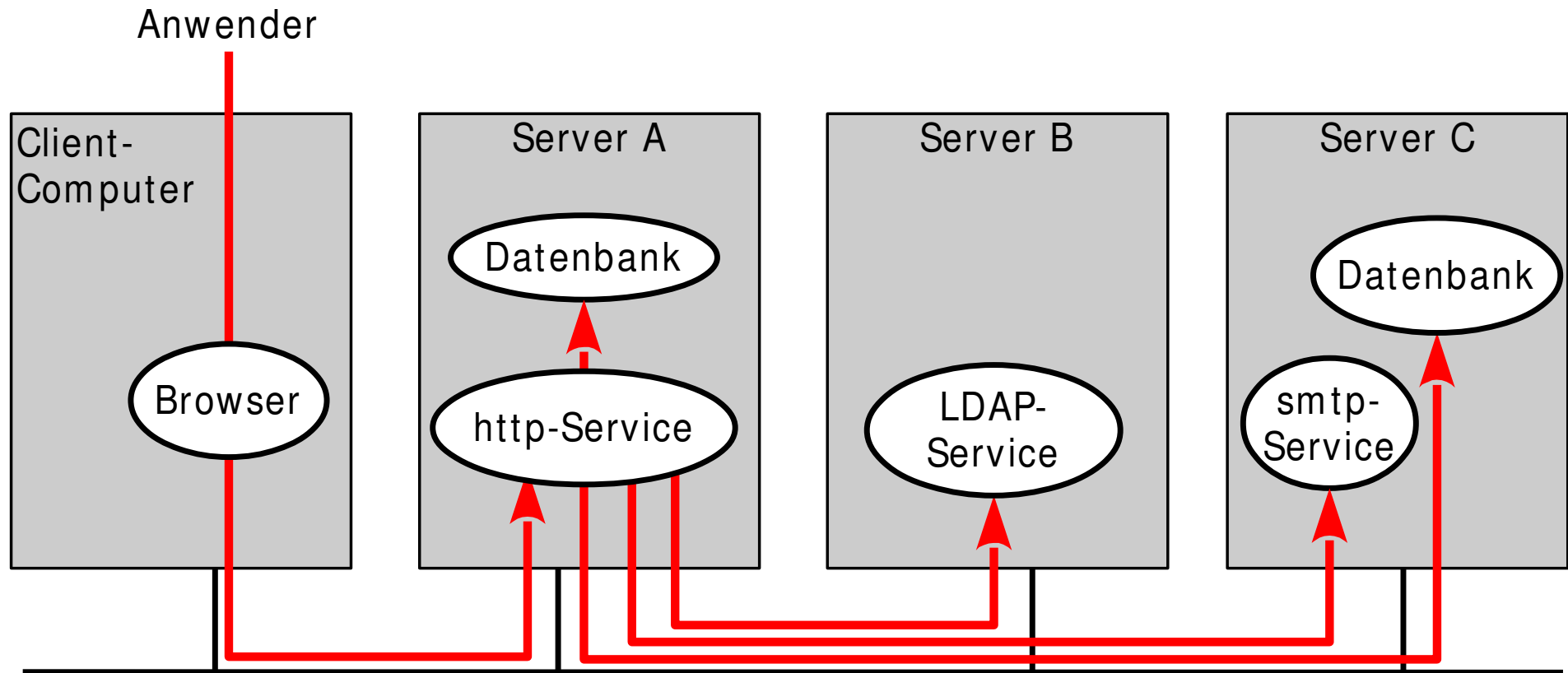
Frontend-Services - Backend-Services

- x Es ist möglich und sinnvoll, dass Services, die dem Anwender zur Verfügung gestellt werden, zur Erbringung des Service andere Services verwenden.
- x Services, mit denen sich das Client-Programm verbindet bezeichnet man als Frontend-Services.
- x Services, die ein Frontendservice zur Erledigung seiner Aufgabe nutzt, bezeichnet man als Backend-Services.
 - ⇒ Bei der Kommunikation zwischen Frontend- und Backendservice ist der Frontend-Service in der Rolle eines Clients.

Backend-Services

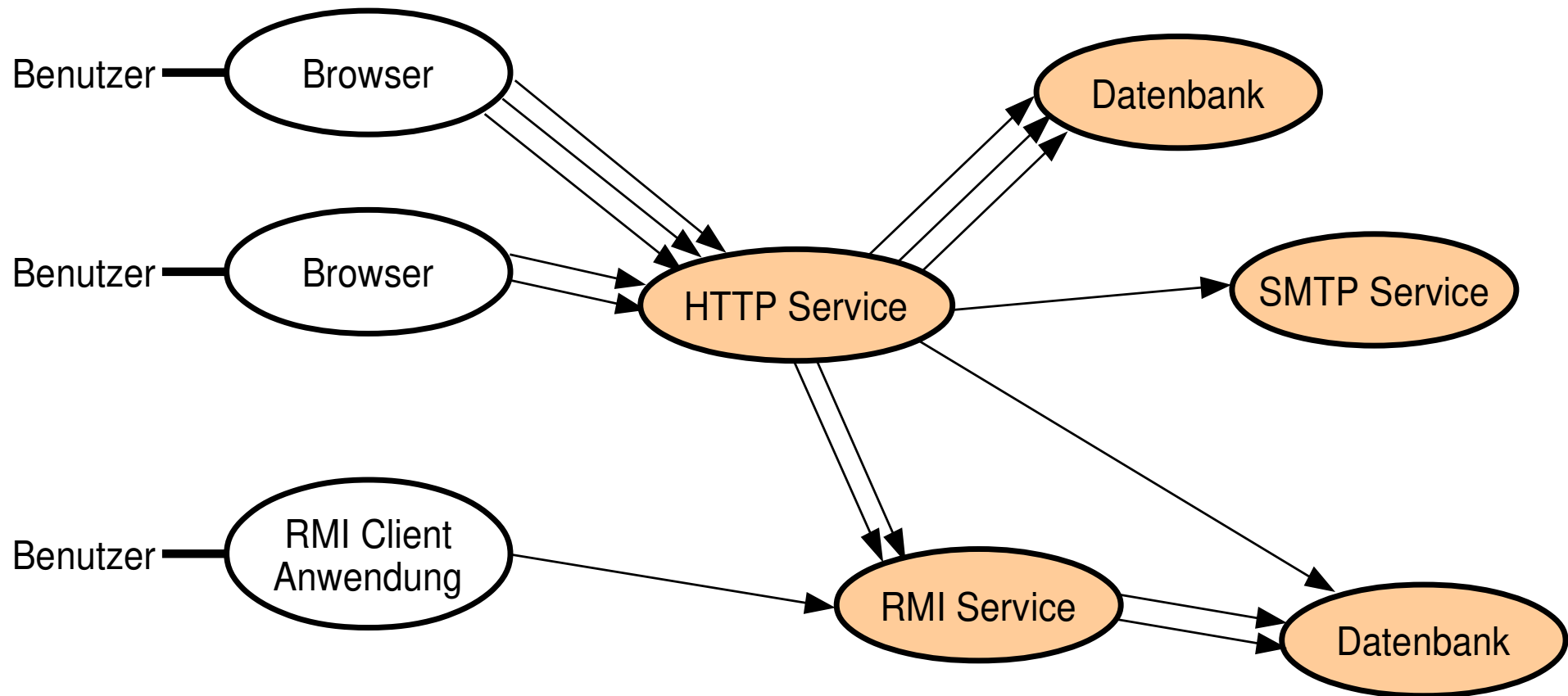
- x Man kann unterscheiden zwischen lokalen Backend-Services, die sich auf dem gleichen Server befinden wie der Frontend-Service und externen Backend-Services.
- x Typische Backend-Services sind Datenbanken, Benutzerverwaltungssysteme, Message-Queueing-Systeme, Transaktionsmanager, E-Mail-Services etc.
- x In der Regel befindet sich der anwendungsspezifische Programmcode nur im Frontend-Service und nicht in den Backend-Services. Die Backend-Services werden lediglich durch Konfiguration an die Anwendung angepasst.

Beispiel



Die roten Pfeile stehen hier für: Client benutzt Service

Verbindungen zwischen Clients und Services



Die Pfeile stehen hier für Verbindungen

3.4 Client/Server-Anwendungen mit Java

Client/Server-Anwendungen und Java

- x Prinzipiell können beliebige Client/Server-Anwendungen mit Hilfe von elementarer Socket-Programmierung realisiert werden.
 - ⇒ Klassen: Socket, ServerSocket
- x Es gibt in Java (und anderen Programmiersprachen) jedoch eine große Anzahl von Client-Libraries und Application Server, mit deren Hilfe verteilte Anwendungen schneller und einfacher umgesetzt werden können.

Client-Libraries

- x In Java gibt es für die verbreiteten Anwendungsprotokolle bereits spezialisierte Bibliotheken, mit deren Hilfe man in komfortabler Weise Client-Programme zu diesen Services schreiben kann.
- x Intern verwenden die Bibliotheken elementare Socket-Operationen.
- x Im einfachsten Fall stellt die Bibliotheken die Dienste des Servers in Form von Methoden zur Verfügung. Der Aufruf einer Dienst-Methoden führt intern dazu,
 - ⇒ dass ein Request in der durch das Anwendungsprotokoll festgelegten Weise kodiert wird
 - ⇒ dass dieser Request an den Server geschickt wird
 - ⇒ dass auf die Response des Servers gewartet wird
 - ⇒ dass die Response ausgewertet und das Ergebnis zurückgegeben wird.

... Client-Libraries

x Beispiele

- ⇒ JDBC (java data base connectivity)
- ⇒ JNDI (java naming and directory interface)
- ⇒ JavaMail (senden/empfangen von E-Mails über verschiedene Protokolle)
- ⇒ HttpURLConnection (http-client-Verbindung)
- ⇒ JMS (java message service)

Application-Server

- x Infrastruktur zur Erstellung eines Services für ein standardisiertes Anwendungsprotokoll.
- x Objektorientiertes Framework
 - ⇒ Das Framework betreibt Objekte einer bestimmten Service-Klasse.
- x Realisierung einer Anwendung
 - ⇒ Implementierung einer oder mehrerer Sohnklassen zur Service-Klasse.
 - ⇒ Die relevanten Methoden werden in geeigneter Weise überschrieben.
 - ⇒ Die Sohnklassen werden dem Application Server zur Verfügung gestellt (Konfiguration)
 - ⇒ Der Application Server wird gestartet.

Aufgaben des Application-Servers

- x Der Application Server ist ein startbares Hauptprogramm (main-Methode).
- x Der Application Server reserviert nach dem Start den für den Dienst benötigten Port (oder auch mehrere Ports) und gibt ihn nach Beendigung wieder frei.
- x Das Verhalten des Application-Server kann über zahlreiche Konfigurationsdaten festgelegt werden. In der Regel befinden sich die Konfigurationsdaten in Dateien. In den Konfigurationsdaten findet der Application-Server auch die Information, welche Service-Klassen er wie betreiben soll.
- x Der Application Server verwaltet einen Thread-Pool und dispatched die eingehenden Verbindungswünsche.

Aufgaben des Application-Servers

- x Sobald eine Verbindung eingeht, reicht der Dispatcher die Requests an die Working-Threads weiter.
- x Die Working-Threads prüfen, welche Service-Klasse für die Bearbeitung des jeweiligen Requests zuständig ist, bilden von der jeweiligen Service-Klasse falls noch nicht vorhanden eine Instanz und rufen dort eine definierte Methode zur Abarbeitung des Requests auf.
- x Das Framework übernimmt die Codierung der Requests und Responses gemäß Anwendungsprotokoll.
- x Die Service-Klasse enthält Methoden, mit deren Hilfe der anwendungsspezifische Programmcode der Sohnklassen auf zentrale Funktionen des Application-Servers zugreifen kann.

Kapitel 4

Datenbankanbindung

Daten speichern

- x Die Aufgabe einer verteilten Anwendung ist es, Daten zu verarbeiten und Sie über eine räumliche und zeitliche Distanz zu transportieren.
 - ⇒ Überwindung der räumlichen Distanz: Netzwerke
 - ⇒ Überwindung der zeitlichen Distanz: Datenspeicher
- x In Java-Programmen werden Daten in Variablen gespeichert.
 - ⇒ Problem: keine Persistenz
 - ⇒ Die Daten befinden sich im Hauptspeicher und gehen nach dem Abschalten des Computers verloren.
 - ⇒ Daten, die Geschäftsvorgänge repräsentieren, sind wertvoll und müssen vor Verlust geschützt werden.

Betriebssicherheit von Services

- x Der Betrieb eines Services kann dadurch gestört werden, dass
 - ⇒ das Service-Programm beendet wird, weil beispielsweise der Server-Computer ausfällt, auf dem es ausgeführt wird
 - ⇒ das Netzwerk ausfällt oder fehlerhaft arbeitet, über das sich die Clients mit dem Service verbinden
 - ⇒ ein Client ausfällt, der sich mit dem Service verbunden hatte
- x Derartige Störungen können nicht ausgeschlossen werden.
- x In all diesen Situationen werden Netzwerk-Verbindungen zwischen Client und Server unterbrochen.
- x Bei diesen Störungen darf angenommen werden, dass sie von temporärer Natur sind, und dass der Betrieb nach der Behebung der Probleme fortgesetzt werden kann und soll.

Störung - Wiederanlauf

Nachdem es zu einer solchen Störung gekommen ist und nachdem die Ursache für die Störung behoben wurde soll der Betrieb wieder aufgenommen werden.

- x Server: Wurde der Service beendet, so wird er wieder gestartet.
- x Client: Nach einem Verbindungsabbruch wird die Verbindung zum Server erneut aufgebaut.

Geschäftsdaten - Geschäftslogik

x Geschäftsdaten (business data)

- ⇒ Daten, die den aktuellen Zustand des Geschäfts repräsentieren.
- ⇒ Beispiele: eingegangene Bestellungen, Benutzerdaten, Reklamationen, Anfragen, abgeschlossene Verträge, Urlaubsanträge, genehmigte Dienstreisen, ausgeliehene Geräte, Kundenkontakte, vereinbarte Termine, ...

x Geschäftslogik (business logic)

- ⇒ Definierte Menge von Transaktionen, mit denen man den Zustand der Geschäftsdaten verändern kann.
- ⇒ Die Geschäftsdaten können nur über diese Transaktionen verändert werden.

... Geschäftsdaten - Geschäftslogik

- x Die Geschäftsdaten und die Geschäftslogik bilden den Kern eines Services.
 - ⇒ Use Cases: Wer soll mit dem Service was machen können?
- x Sinn und Zweck eines Services ist es, dem Anwender die Möglichkeit zu verschaffen, bestimmte Geschäftstransaktionen ausführen und den aktuellen Zustand der Geschäftsdaten abfragen zu können.
- x Zu vorgegebenen Geschäftsdaten und zu einer vorgegebenen Geschäftslogik können unterschiedliche Formen von Client-Service-Anwendungen konstruiert werden. Beispiele:
 - ⇒ Web-Anwendung
 - ⇒ Client-Server-Anwendung mit App als Client-Anwendung
 - ⇒ Client-Server-Anwendung mit Desktop-Anwendung als Client
 - ⇒ ...

Persistenz

- x Der in der Datenbank gespeicherte Zustand der Daten muss dort dauerhaft zur Verfügung stehen und darf nicht verloren gehen.
- x Zur persistenten Speicherung eignen sich Datenspeicher, die auch dann ihren Zustand behalten, wenn sie nicht mit einer Stromversorgung verbunden sind: Festplatten, Flash-Speicher.
- x Aber auch diese „persistenten Datenspeicher“ haben eine begrenzte Lebensdauer. Es gibt keine Geräte, mit denen Daten „auf ewig“ gespeichert werden können.
- x Es gibt keine absolute Persistenz. Persistenz muss graduell betrachtet werden. Man muss ein hohes Maß an Verlostsicherheit anstreben.
- x Maßnahmen zur Verbesserung der Persistenz:
 - ⇒ regelmäßige Backups
 - ⇒ Spiegelung der Daten
- x Der Service garantiert, dass die Geschäftsdaten auch nach einer Störung noch zur Verfügung stehen und dass es durch die Störung nicht zu inkonsistenten Zuständen in den Geschäftsdaten kommen kann.

Atomizität

- x Atomizität: Transaktionen sollen ganz oder gar nicht ausgeführt werden.
- x Nach einer erfolgreichen Ausführung einer Transaktion, soll das System den erfolgreichen Abschluss der Transaktion melden. Der Anwender soll von nun an davon ausgehen können, dass die Änderungen auf den Daten vollständig vorgenommen wurden und dass die Änderungen persistent gespeichert werden.
- x Eine Transaktion setzt sich intern aus mehreren Teilschritten zusammen.
- x Störungen während der Ausführung der Transaktionen können nicht ausgeschlossen werden.
- x Störungen während der Transaktionsausführung sollen dazu führen, dass die bereits ausgeführten Teilschritte rückabgewickelt werden. Eine gescheiterte Transaktion bewirkt keine Änderungen bei den Geschäftsdaten.

Nebenläufiger Zugriff auf Geschäftsdaten

- x In der Regel sind Services nebenläufig realisiert.
 - ⇒ Mehrere Clients können sich gleichzeitig mit dem Service verbunden.
 - ⇒ Der Service wird durch mehrere Threads realisiert, wobei jeder Thread eine Verbindung bearbeitet.
 - ⇒ Jeder der Threads soll auf die Geschäftsdaten zugreifen können.
- x Forderung
 - ⇒ Auf die Geschäftsdaten muss von mehreren Clients gleichzeitig in nebenläufiger Weise zugegriffen werden können.
 - ⇒ gleichzeitig mehrere Geschäftstransaktionen und mehrere Leseoperationen auf den Geschäftsdaten

4.1 Relationale Datenbanken

Relationale Datenbanken

- x Relationale Datenbank: Service zur Speicherung von Daten
- x Relationale Darstellung der Daten
 - ⇒ Daten werden in Tabellenform gespeichert (endliche Relationen)
 - ⇒ Standard für den Zugriff auf relationale Datenbanken: SQL (structured query language)
 - ⇒ Definition von Tabellen, Sichten und Konsistenzbedingungen über SQL-DDL (data definition language).
 - ⇒ Einfügen, löschen und ändern von Einträgen (Zeilen) in Tabellen über SQL-DML (data manipulation language).
- x Der Datenbank-Dienst kann über unterschiedliche Programmierschnittstellen von Clients angesprochen, über die diese SQL-Anfragen an die Datenbank schicken.

... Relationale Datenbanken

x Persistenz

- x* Datenbanken garantieren Persistenz. Sie verwenden zur Speicherung der Daten persistente Speichermedien.
- x* Daten sind auch nach dem Abschalten des Computers noch verfügbar.
- x* Verschiedene Mechanismen können vom Administrator der Datenbank konfiguriert werden, um ein möglichst hohes Maß an Persistenz zu erzielen: Backups, Spiegelungen, etc.

x Atomizität der Transaktionen

- ⇒ Änderungen der Daten erfolgen durch Transaktionen. Die Datenbank garantiert, dass Transaktionen immer vollständig oder aber gar nicht ausgeführt werden.

... Relationale Datenbanken

x Konsistenz *

- ⇒ Die Datenbank garantiert die Konsistenz der Datenstrukturen: die Einträge in den Tabellen enthaltenen Daten stimmen mit den in der Tabellendeklaration vorgegebenen Datentyp überein.
- ⇒ Für die in SQL-DDL definierten Datenstrukturen können explizit zusätzliche Konsistenzbedingungen vereinbart werden:
 - x* Schlüsselintegrität (Eigenschlüssel)
 - x* referenzielle Konsistenz (Fremdschlüssel)
 - ...

* Andere Bezeichnung: Integrität

... Relationale Datenbanken

x Nebenläufiger Zugriff

- ⇒ Eine Datenbank kann von mehreren Clients gleichzeitig genutzt werden.
- ⇒ Die Nutzung der Datenbank erfolgt verbindungsorientiert: jeder Client baut eine Verbindung zu der Datenbank auf, um über die Datenbankverbindung Datenbankoperationen auszuführen.

x Rechteverwaltung: Authentifizierung, Autorisierung

- ⇒ Von den Clients, die auf die Datenbank zugreifen wollen, kann beim Verbindungsaufbau gefordert werden, dass sie sich authentifizieren.
- ⇒ In Datenbank kann in einer differenzierten Weise festgelegt werden, welche Benutzer wie auf welche Daten zugreifen dürfen. (Autorisierung)

Datenbanken in Services verwenden

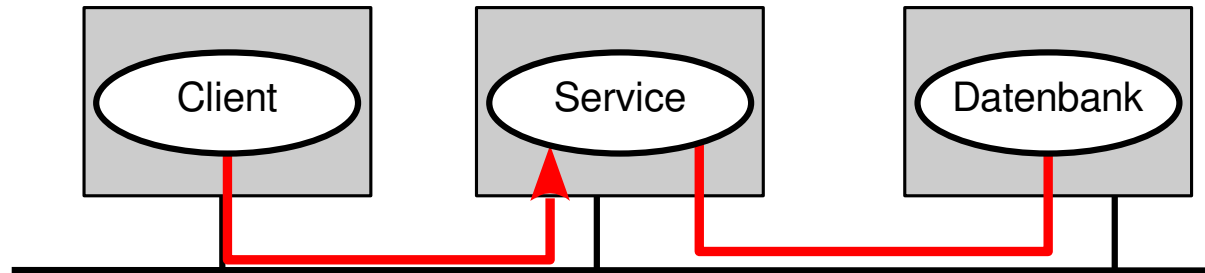
- x Datenbanken bieten eine gute Möglichkeit zur Aufbewahrung von Geschäftsdaten, die innerhalb eines Services verwendet werden sollen. Sie erfüllen insbesondere die folgenden drei Forderungen:
 - ⇒ Persistenz der Daten
 - ⇒ Atomizität der Transaktionen
 - ⇒ nebenläufiger Zugriff
- x Datenbanken stellen selbst einen Service dar, der entweder lokal über Prozeduraufrufe oder entfernt über eine Netzwerkverbindung genutzt werden kann. (Backend-Service)

Geschäftsdaten in Datenbanken

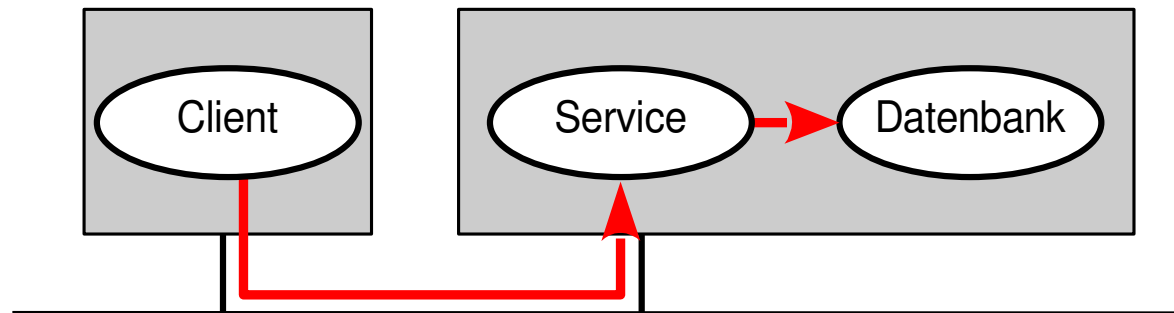
- x Die Geschäftsdaten müssen auf Datenbankstrukturen abgebildet werden: Tabellen, Schlüsseln, Sichten, Fremdschlüsseln etc.
- x Die Geschäftsdaten müssen zu Beginn des Betriebs der Anwendung auf einen definierten Anfangszustand gesetzt werden.
- x Create-And-Populate-Skript
 - ⇒ Datei mit SQL-Befehlen
 - ⇒ SQL-DDL-Befehle zur Erzeugung der benötigten Strukturen (create-Befehle)
 - ⇒ insert-Operationen zur Befüllung der Tabellen mit initialen Werten (populate)

Datenbanken: extern, lokal, eingebettet

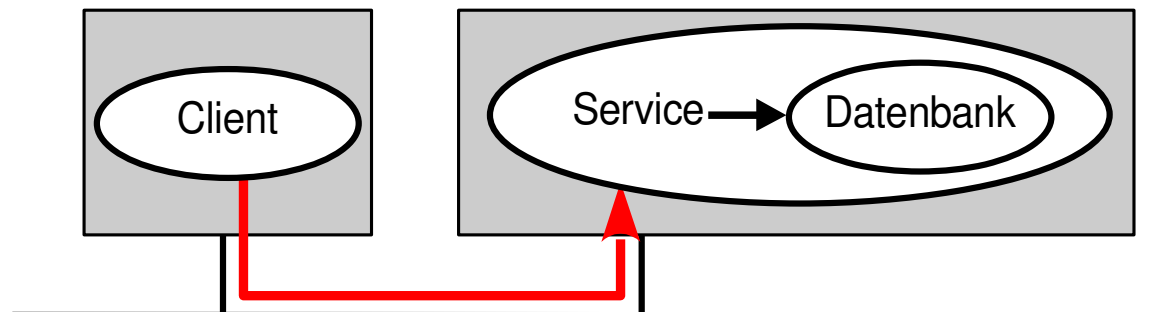
externe Datenbank



lokale Datenbank



eingebettete Datenbank



Datenbanken: extern, lokal, eingebettet

x extern

- ⇒ Die Datenbank befindet sich auf einem anderen Computer als der Service, der sie benutzt.
- ⇒ Zugriff über das Netzwerk

x lokal

- ⇒ Die Datenbank befindet sich auf dem selben Computer wie der Service, der sie benutzt.
- ⇒ Zugriff per Netzwerk-Protokoll innerhalb des Computers (localhost, 127.0.0.1)

x eingebettet

- ⇒ Die Datenbank ist Teil des Service-Programms.
- ⇒ Die eingebettete Datenbanken wird als Library in das Programm eingebunden.
- ⇒ Der Service verwendet Methodenaufrufe um SQL-Befehle an die Datenbank zu schicken. Keine Netzwerkkommunikation.

4.2 Elementare SQL-Befehle

Vorbemerkung

Ein SQL-Befehl ist eine Zeichenkette, die sich aus mehreren Token (Teilstücken) zusammensetzt, die durch Leerzeichen voneinander getrennt sind. Es gibt drei Arten von Tokens:

- x Schlüsselwörter der Sprache SQL. Beispiele: CREATE, WHERE,...
⇒ keine Umklammerung
- x Vom Anwender der Datenbank definierte Bezeichner (Namen) für Tabelle, Spalten etc.
⇒ Umklammerung mit ` ... `
- x Datenwerte, die in das Feld eines Datensatzes geschrieben werden sollen oder die mit einem solchen Feld verglichen werden sollen.
⇒ Umklammerung ' ... '

Create Table

```
CREATE TABLE `studenten` (  
  `vorname` varchar(30),  
  `nachname` varchar(30),  
  `matrikelnummer` int,  
  `geburtsdatum` date  
)
```

studenten

vorname	nachname	matrikelnummer	geburtsdatum
---------	----------	----------------	--------------

... *Create Table*

- x Dieser Befehl erzeugt eine Tabelle mit dem Namen *studenten*. In der Tabelle *studenten* werden Datensätze gespeichert, wobei jeder dieser Datensätze aus vier Feldern besteht: *vorname*, *nachname*, *matrikelnummer* und *geburtsdatum*.
- x In der Tabellendarstellung entsprechen die Spalten den Feldern. In der Kopfzeile stehen die Namen der Felder. Die Zeilen unterhalb der Kopfzeile stehen für die Datensätze.
- x Nach dem Erzeugen der Tabelle ist die Tabelle leer, sie enthält noch keine Datensätze.
- x Jedes Feld hat einen Namen und einen Typ. In SQL gibt es spezielle Typen, die sich von den Typen klassischer Programmiersprachen unterscheiden.
 - ⇒ *varchar(30)* steht beispielsweise für eine Zeichenkette mit nicht mehr als 30 Zeichen
 - ⇒ *int* steht für eine ganze Zahl in Zweierkomplementkodierung mit 32 Bit. Das entspricht in Java dem Datentyp *int*.
 - ⇒ *date* steht für ein Datum. Das entspricht in Java dem Datentyp *Date*.

Drop Table

```
DROP TABLE `studenten`
```

Lösche die Tabelle *studenten*!

Insert

```
INSERT INTO `studenten`  
  (`vorname`, `nachname`, `matrikelnummer`, `geburtsdatum`)  
VALUES  
  ('Lea', 'Maier', '88236', '2002-07-05'),  
  ('Horst', 'Müller', '78665', '2004-09-05'),  
  ('Lisa', 'Müller', '78334', '2004-09-05'),  
  ('Otto', 'Zimmermann', '13278', '2002-12-09')
```

studenten

vorname	nachname	matrikelnummer	geburtsdatum
Lea	Maier	88236	2001-07-05
Horst	Müller	78665	2004-09-05
Lisa	Müller	78334	2004-09-05
Otto	Zimmermann	13278	2002-12-09

Insert

- x Der Befehl *insert* legt zunächst fest, in welche Tabelle etwas eingefügt werden soll. In diesem Fall: In die Tabelle *studenten*.
- x Dann wird festgelegt, wie die Datensätze aussehen sollen, die eingefügt werden. In diesem Fall wird angegeben, dass die einzufügenden Datensätze aus allen Feldern der Tabelle bestehen sollen: *vorname*, *nachname*, *matrikelnummer* und *geburtsdatum*. Es folgen mehrere Datensätze, die diesen Aufbau haben.
- x Hinweis: Man könnte beim Einfügen von Datensätzen ein paar Felder der Tabelle weglassen. Fügt man einen Datensatz ein, bei dem ein Feld fehlt, so wird für dieses Feld ein vordefinierter Defaultwert verwendet.

Tabellen sind Multimengen von Datensätzen

- x In der Mathematik steht der Begriff Menge für eine ungeordnete Ansammlung von Elementen. Jeder Gegenstand der Betrachtung kann in einer Menge enthalten sein oder eben nicht.
- x Multimengen sind anders. In einer Multimenge kann ein Gegenstand der Betrachtung mehrfach enthalten sein. Für jedes Element einer Multimenge kann angegeben werden, wie oft es in der Multimenge enthalten ist.
- x Eine SQL-Tabelle steht für eine Multi-Menge von Datensätzen. Innerhalb der Tabelle haben die Datensätze keine Reihenfolge.
 - ⇒ In einem Tabellenkalkulationsprogramm wie Excel haben die Zeilen hingegen eine definierte Reihenfolge.

Select

```
SELECT * FROM `studenten` WHERE `nachname` = 'Müller'
```

vorname	nachname	matrikelnummer	geburtsdatum
Horst	Müller	78665	2004-09-05
Lisa	Müller	78334	2001-09-05

Gib mir von der Tabelle *studenten* die Datensätze, bei denen *nachname* den Wert Müller hat!

Select

- x Mit dem Befehl *select* wird lesend auf Datensätze zugegriffen. Man erhält von der Tabelle eine Teilmenge der Datensätze und bei diesen Datensätzen kann man auch noch festlegen, welche Felder der Datensätze man gerne hätte.
- x Das Sternchen * steht dafür, dass von den Datensätzen alle Felder ausgelesen werden sollen. Möchte man nur auf ganz bestimmte Felder zugreifen, so kann anstelle des Sternchens eine Liste mit den Feldnamen angeben, auf die man zugreifen möchte.
- x Der nachfolgende Befehl wählt wieder genau die Studenten aus, deren Nachname *Müller* ist, liest von diesen Datensätzen jedoch nur die Felder *vorname* und *geburtsdatum* aus.

```
SELECT `vorname`, `geburtsdatum` FROM `studenten` WHERE `nachname` = 'Müller'
```

vorname	geburtsdatum
Horst	2004-09-05
Lisa	2004-09-05

Der where-Abschnitt

- x Der *where*-Abschnitt steht für einen Filter. Hier steht ein boolescher Ausdruck (eine Eigenschaft), den all Datensätze erfüllen sollen, die man ansprechen möchte.
- x Ein *select*-Befehl ohne *where*-Abschnitt liest alle Datensätze.
- x Eine einfacher boolescher Ausdruck ist ein Vergleich zweier Terme. Terme sind entweder Felder oder Konstanten oder arithmetische Ausdrücke. In den arithmetischen Ausdrücken können die Grundrechenarten + - * / verwendet werden. Vergleichsoperatoren sind = (Gleichheit), < (kleiner als) und > (größer als).
- x Mit den Vergleichsoperatoren < und > können nicht nur Zahlenwerte verglichen werden, sondern auch Zeichenketten, Datumswerte und Zeitangaben. Bei Zeichenketten stehen die Operatoren für die alphabetische Ordnung der Zeichenketten. Bei Datumswerten und Zeitangaben prüfen diese Vergleiche, ob der beschriebene Zeitpunkt vor oder nach dem anderen Zeitpunkt steht.
- x Einfache boolesche Ausdrücke können mit logischen Operatoren verknüpft werden. Boolesche Operatoren sind: *AND*, *OR*, *NOT*.

Ergebnisliste - Tabelle

```
SELECT `nachname`, `geburtsdatum` FROM `studenten` WHERE `nachname` = 'Müller'
```

nachname	geburtsdatum
Müller	2004-09-05
Müller	2004-09-05

- x Das Ergebnis einer *select*-Abfrage ist eine Liste von Datensätzen. Die Elemente in dieser Liste haben eine definierte Reihenfolge. Die Liste kann ein und den selben Datensatz mehrfach enthalten.
- x Gerne wird das Ergebnis einer *select*-Abfrage als „result set“ bezeichnet. Diese Bezeichnung ist genau betrachtet falsch. Eine Menge kann Elemente nicht mehrfach enthalten. Außerdem haben die Datensätze in dem „result set“ eine definierte Reihenfolge, die Datensätze in einer Tabelle dagegen nicht.

Select Distinct

```
SELECT DISTINCT `nachname`, `geburtsdatum` FROM `studenten`  
WHERE `nachname` = 'Müller'
```

nachname	geburtsdatum
Müller	2004-09-05

- x Der *select-distinct*-Befehl erzeugt eine Liste von Datensätzen, in der jeder Datensatz nur einmal vorkommt.
- x „Dopplungen werden entfernt“
- x Bei einer *select-distinct*-Abfrage kann die Ergebnisliste tatsächlich als eine Menge betrachtet werden.

Update

```
UPDATE `studenten`  
  SET `geburtsdatum` = '2004-09-15'  
  WHERE `matrikelnummer` = 78665
```

studenten

vorname	nachname	matrikelnummer	geburtsdatum
Lea	Maier	88236	2001-07-05
Horst	Müller	78665	2004-09-15
Lisa	Müller	78334	2004-09-05
Otto	Zimmermann	13278	2002-12-09

Nimm die Tabelle *studenten* und setze dort das Feld *geburtsdatum* auf den Wert *2004-09-15* und zwar bei all den Datensätzen, bei denen *matrikelnummer* den Wert 78665 hat!

Update

- x Nach dem Namen der Tabelle steht eine Zuweisung: Ein bestimmtes Feld bekommt einen bestimmten Wert zugewiesen.
- x Diese Änderung des Wertes soll bei all den Datensätzen angewandt werden, die nachfolgend beschrieben werden. Welche Datensätze von der Änderung betroffen sein sollen, legt der *where*-Abschnitt fest. Der *where*-Abschnitt des *update*-Befehls hat den gleichen Aufbau wie der des *select*-Befehls.
- x In dem Beispiel war nur ein Datensatz von der Änderung betroffen, es können aber auch mehrere Datensätze Änderungen betroffen sein. Fehlt der *where*-Abschnitt, so werden diese Änderungen auf alle Datensätze angewandt.
- x In dem *set*-Abschnitt können auch mehrere Zuweisungen stehen. Dann werden bei der Ausführung des Befehls all diese Zuweisungen auf die betroffenen Datensätze angewandt.

Delete

```
DELETE FROM `studenten`  
WHERE `matrikelnummer` < 78500
```

studenten


vorname	nachname	matrikelnummer	geburtsdatum
Lea	Maier	88236	2001-07-05
Horst	Müller	78665	2004-09-15
Lisa	Müller	78334	2004-09-05
Otto	Zimmermann	13278	2002-12-09

Lösche in der Tabelle *studenten* all die Datensätze, bei denen *matrikelnummer* kleiner als *78500* ist!

Primärschlüssel

```
ALTER TABLE `studenten` ADD PRIMARY KEY( `matrikelnummer`);
```

studenten

vorname	nachname	matrikelnummer 	geburtsdatum
Lea	Maier	88236	2001-07-05
Horst	Müller	78665	2004-09-05
Lisa	Müller	78334	2004-09-05
Otto	Zimmermann	13278	2002-12-09

Ändere die Tabelle *studenten* derart ab, dass das Feld *matrikelnummer* zum Primärschlüssel wird.

Primärschlüssel

- x Wenn eine Feld ein Primärschlüssel ist, dann sorgt die Datenbank dafür, dass die Tabelle zu keinem Zeitpunkt zwei Datensätze mit dem gleichen Primärschlüssel enthält.
- x Daraus folgt, dass es in dieser Tabelle zu keinem Zeitpunkt zwei Datensätze mit der selben Matrikelnummer gibt.
- x Für jede denkbare Nummer gibt es in der Tabelle somit entweder einen oder keinen Datensatz mit dieser Matrikelnummer.
- x Ein *where*-Abschnitt der Bauart

```
WHERE `matrikelnummer` = 12345
```

hat nur zwei mögliche Ergebnisse:

- ⇒ Die Ergebnisliste enthält genau einen Datensatz: den Datensatz mit dieser Matrikelnummer.
- ⇒ Die Ergebnisliste ist leer. Es gibt zu dieser Matrikelnummer keinen Datensatz.

Primärschlüssel

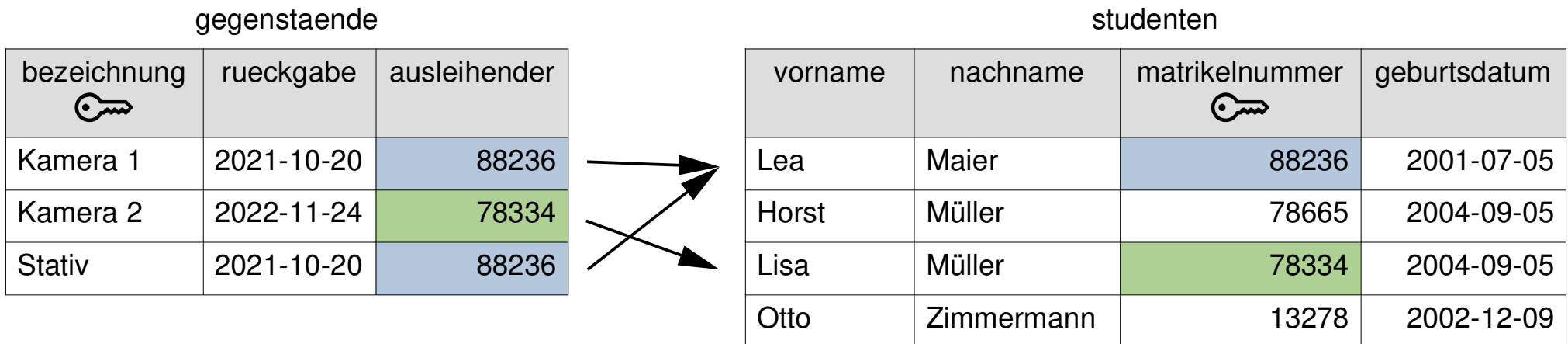
- x Ein Primärschlüssel definiert eine Einschränkung (Constraint).
- x Eine *insert*-Operation scheitert, wenn ein einzufügender Datensatz im Primärschlüsselfeld einen Wert hat, den es in der Tabelle bereits gibt.
- x Eine *update*-Operation update scheitert, wenn dem Primärschlüssel ein Wert zugewiesen werden soll, den es bei einem anderen Datensatz bereits gibt.

Lookup, Indizes

- x Hat ein Datensatz einen Primärschlüssel, so dient dieses Feld als eindeutiger Identifikator für den Datensatz. Der Primärschlüssel verweist auf den Datensatz.
 - ⇒ Objekte in Java
 - ⇒ Verweise sind Adressen im Hauptspeicher und beschreiben, wo sich das Objekt im Hauptspeicher befindet.
 - ⇒ Primärschlüssel in SQL
 - ⇒ Primärschlüssel verweisen auf einen Datensatz in einer Tabelle. Mit *select*-, *update*- und *delete*-Befehlen kann der Datensatz angesprochen werden, jeweils mit einem *where*-Abschnitt der Bauart:
WHERE primärschlüssel = xy
- x Das Nachschlagen eines Datensatzes mit Hilfe dessen Primärschlüssel (Lookup) ist eine Aufgabe, die eine Datenbank sehr häufig durchführen muss. Von der Datenbank wird erwartet, dass sie diese Aufgabe sehr schnell abarbeiten kann und zwar selbst dann, wenn die Anzahl der Datensätze sehr groß ist.
- x Dazu baut die Datenbank im Hintergrund zu den Primärschlüsseln eigene Datenstrukturen auf, die als Indizes bezeichnet werden.

Fremdschlüssel

Hat eine Tabelle einen Primärschlüssel, so kann man in einem Datensatz einer anderen Tabelle auf einen Datensatz dieser Tabelle verweisen, indem man dort in einem Feld den Wert des Primärschlüssels speichert. Ein Feld, das auf einen anderen Datensatz verweist, nennt man einen Fremdschlüssel.



Referenzielle Integrität

```
ALTER TABLE `gegenstaende`  
  ADD CONSTRAINT `ausleihender`  
  FOREIGN KEY (`ausleihender`) REFERENCES `studenten`(`matrikelnummer`)
```

Ändere die Tabelle *gegenstaende* derart, dass von jetzt an das Feld *ausleihender* (Fremdschlüssel) immer auf genau einen Datensatz der Tabelle *studenten* verweist, wobei in diesem Datensatz im Feld *matrikelnummer* (Primärschlüssel) der gleiche Wert steht wie im Fremdschlüssel.

Referenzielle Integrität

- x Referenzielle Integrität bedeutet, dass zu jedem Zeitpunkt jeder der Fremdschlüssel auf einen Datensatz in der Zieltabelle verweist.
- x Mit dem Befehl
ALTER TABLE...ADD CONSTRAINT...FOREIGN KEY
wird ein Constraint (eine Einschränkung) für einen Fremdschlüssel definiert. Der Fremdschlüssel-Constraint legt fest, dass die Datenbank für referenzielle Integrität bezüglich des angegebenen Fremdschlüssels und des Primärschlüssels in der Zieltabelle sorgen soll.

Referenzielle Integrität

Es gibt mehrere Gestaltungsmöglichkeiten, wenn man einen Fremdschlüssel-Constraint definiert.

Der Fremdschlüssel-Constraint in dem Beispiel steht für den Standardfall. Standardmäßig sorgt die Datenbank dafür, dass SQL-Operationen scheitern, wenn sie dazu führen würden, dass die referenzielle Integrität verletzt wird.

- x Ein Datensatz mit einem Fremdschlüsselfeld kann nicht eingefügt werden, wenn der Fremdschlüssel nicht auf einen Datensatz in der Zieltabelle verweist.
- x Ein Fremdschlüsselfeld kann nicht so verändert werden, dass der Fremdschlüssel nach der Änderung nicht mehr auf einen Datensatz verweist.
- x Ein Datensatz kann nicht gelöscht werden, wenn es einen Datensatz mit einem Fremdschlüssel gibt, der auf diesen Datensatz verweist.
- x Der Primärschlüssel eines Datensatzes kann nicht geändert werden, wenn es einen Datensatz mit einem Fremdschlüssel gibt, der auf diesen Datensatz verweist.

Referenzielle Integrität

Es gibt Varianten von Fremdschlüssel-Constraints, bei denen SQL-Operationen in diesen Situationen nicht scheitern, sondern Änderungen an benachbarten Datensätzen vornehmen, um auf diese Weise die referenzielle Integrität sicherzustellen.




- x Wenn ein Datensatz gelöscht wird, dann lösche auch alle Datensätze, die auf ihn verweisen (Cascading-Delete).
- x Wenn ein der Primärschlüssel eines Datensatzes geändert wird, dann ändere in gleicher Weise alle Fremdschlüsselfelder, die auf den Datensatz verweisen.
- x ...

Primärschlüssel mit mehreren Feldern

- x Ein Feld eignet sich nicht als Primärschlüssel, wenn mehrere Datensätze gespeichert werden sollen, bei denen dieses Feld den gleichen Wert hat.
 - ⇒ Das Feld *nachname* eignet sich bei der Tabelle *studenten* nicht, da potenziell mehrere Datensätze gleichen Nachnamens gespeichert werden sollen.
- x Oft kommen unterschiedliche Felder als Primärschlüssel in Betracht.
 - ⇒ In einer Tabelle mit Studenten, in der man pro Student sowohl die Matrikelnummer als auch die Personalausweisnummer speichert, kämen sowohl die Matrikelnummer als auch die Personalausweisnummer als Primärschlüssel in Betracht.

Primärschlüssel mit mehreren Feldern

- x In manchen Fällen muss man mehrere Felder kombinieren, um Datensätze in einer Tabelle eindeutig identifizieren zu können.
 - ⇒ In einer Tabelle mit allen Räumen der Hochschule Furtwangen setzt sich der Primärschlüssel aus drei Teilen zusammen: Campus, Gebäude, Raumnummer

raeume				
campus 	gebaeude 	raumnummer 	sitzplaetze	Projektoren
Furtwangen	C	0.02	57	1
Furtwangen	C	3.14	18	0
Tuttlingen	B	0.02	38	1

```
ALTER TABLE `raeume`  
  ADD PRIMARY KEY( `campus`, `gebaeude`, `raumnummer`)
```

- x Besteht der Primärschlüssel aus mehreren Feldern, so benötigt man ebendiese Felder auch für den Fremdschlüssel.

SQL-DDL und SQL-DML

- x SQL-DDL: Data Definition Language
 - ⇒ create table, alter table, drop table
 - ⇒ Datenstrukturen für die Datensätze aufbauen
 - ⇒ Aufrufe vor oder zu Beginn des Betriebs der Anwendung
 - ⇒ langsame Ausführung – nicht auf Performance optimiert
- x SQL-DML: Data Manipulation Language
 - ⇒ insert, update, delete, select
 - ⇒ mit den Datensätzen arbeiten
 - ⇒ Aufrufe während des Betriebs der Anwendung
 - ⇒ schnelle Ausführung – die Leistungsfähigkeit der Datenbank bemisst sich an der Ausführungsgeschwindigkeit der DML-Befehle

CRUD-Operationen

- x Datensätze findet man in der IT an verschiedenen Stellen.
 - ⇒ in Datenbanken (Zeilen einer Tabelle)
 - ⇒ in Programmiersprachen (Objekte)
 - ⇒ in Formularen von Webseiten
 - ⇒ ...
- x Ganz allgemein gilt,
 - ⇒ dass man Datensätze erzeugen kann ("create"),
 - ⇒ dass man auf bestehende Datensätze lesend zugreifen kann ("read"),
 - ⇒ dass man den Inhalt eines bestehenden Datensatzes ändern kann ("update") und
 - ⇒ dass man Datensätze löschen kann ("delete")
- x Der Begriff CRUD-Operationen beschreibt in abstrakter Weise diese elementaren Operationen auf Datensätzen.
 - C = create
 - R = read
 - U = update
 - D = delete

CRUD-Operationen in SQL und Java

In SQL sind die elementaren Datensätze Zeilen von Datenbanken. Die CRUD-Operationen sind in SQL gerade die DML-Befehle.

CRUD	SQL
create	insert
read	select
update	update
delete	delete

In Java werden Objekte verwendet, um darin Datensätze aufzubewahren.

CRUD	Java-Objekte
create	Konstruktoraufruf
read	lesender Zugriff auf Objektattribute
update	Wertzuweisung zu Objektattribut
delete	implizites Löschen durch Garbage Collector

4.3 JDBC

JDBC

- x JDBC: Java Data Base Connectivity
- x Java-API für Zugriff auf SQL-basierte, relationale Datenbank
- x Funktionen:
 - ⇒ Herstellen einer Verbindung zu einer Datenbank inklusive Authentifizierung
 - ⇒ Ausführung von Datenbanktransaktionen auf der Basis von SQL

JDBC-Treiber

- x Mit JDBC kann mit einer einheitlichen API auf Datenbanken unterschiedlicher Anbieter zugegriffen werden (Oracle DB, DB2, MySql, MariaDB, Java DB, etc.).
- x JDBC ist in der Standard-Library enthalten, die in jeder JVM enthalten ist.
- x Sobald man mit einer konkreten Datenbank arbeiten möchte, benötigt man für diese Datenbank einen JDBC-Treiber.
- x Der JDBC-Treiber ist eine Java-Library (jar-Datei). Damit der Treiber verwendet werden kann, muss er in den Java-Classpath aufgenommen werden.

...JDBC-Treiber

- x Die Funktionalität, die JDBC zur Verfügung stellt, wird nicht von allen Anbietern von JDBC-Treibern in vollem Umfang unterstützt.
 - x Beispiel: Verwendet man eine ältere Version von MySql, so unterstützt diese kein Transaktionen. Die transaktionsbezogenen Methoden in JDBC können dann nicht benutzt werden.
- x JDBC stellt dem Anwender eine Menge von Klassen zur Verfügung: DataSource, Connection, Statement etc.. Von diesen Klassen findet man in den JDBC-Treibern Sohnklassen mit den datenbankspezifischen Implementierungen.

...JDBC-Treiber

- x Der Anwender sollte sein Programm so schreiben, dass es unabhängig von einer konkreten Datenbank wird und dass die Datenbank und der Datenbanktreiber später ohne Änderungen im Programmcode gewechselt werden kann. Deshalb sollte der Anwender nur die JDBC-Klassen verwendet, nicht aber die Klassen aus den Treibern.
 - ⇒ Es soll in diesem Kapitel noch genauer erläutert werden, wie man dieses Ziel erreicht.

Unterschiedliche Arten von JDBC-Treiber

- x JDBC-Treiber können auf sehr unterschiedliche Weise realisiert werden.
- x Die JDBC-Treiber unterscheiden sich in der Performance und in Kompatibilität bezüglich der Betriebssysteme.
- x Manche JDBC-Treiber enthalten Native Code (Maschinenprogramm-Anteile). Dann können diese Treiber nur auf den jeweils passenden Betriebssystemen ausgeführt werden.
 - ⇒ Die Verwendung von Native Code dient der Performance-Optimierung.
- x Andere JDBC-Treiber bestehen zu 100% aus Java-Byte-Code. Diese Treiber können überall (in jeder JVM) eingesetzt werden.

DataSource

- x Ein Objekt vom Typ DataSource repräsentiert einen Verweis auf eine Datenbank. Es beinhaltet alle Informationen, die dazu benötigt werden, auf eine Datenbank zuzugreifen. Dazu gehören im allgemeinen
 - ⇒ der URL, unter der die Datenbank zu finden ist
 - ⇒ der Benutzernamen und das Passwort zur Authentifizierung
- x Ein DataSource-Objekt enthält die Objektmethode *getConnection()*. Mit dieser Methode baut das Anwendungsprogramm eine Verbindung zur Datenbank auf und führt dann über diese Verbindung SQL-Anweisungen aus.

Erzeugen eines DataSource-Objekts

```
MysqlDataSource mdatasource = new MysqlDataSource();  
mdatasource.setURL("jdbc:mysql://webapp.dm.fh-furtwangen.de/op");  
mdatasource.setUser("op");  
mdatasource.setPassword("geheim");  
DataSource datasource = (DataSource)mdatasource;
```

Arbeiten mit einem DataSource-Objekt

Verbindungsaufbau, SQL-Query abschicken, Verbindungsabbau

```
try (Connection connection = datasource.getConnection(); {  
    Statement statement = connection.createStatement();  
    ResultSet resultSet = statement.executeQuery("select * from studenten");  
    while (resultSet.next()) {  
        System.out.println(resultSet.getString("vorname"));  
        System.out.println(resultSet.getString("nachname"));  
        System.out.println(resultSet.getInt("matrikelnummer"));  
    }  
}
```

Datenbankanbindung

- x Datenbanken sind von Computer zu Computer unterschiedlich realisiert und unterscheiden sich in Datenbank-Typ, URL, User-Id und Passwort.
- x Forderung: Ein Programm soll so implementiert werden, dass es ohne Änderungen im Programmcode an eine beliebige Datenbank angebunden werden kann.
 - ⇒ Bereitstellung eines zusätzlichen JDBC-Treibers für die zu verwendende Datenbank
 - ⇒ Bereitstellung der Konfigurationsdaten: URL, User-ID, Passwort

4.4 Objektrelationales Mapping

Objektrelationales Mapping

- x In Java-Programmen werden die Daten in Objekten gespeichert. Zur persistenten Speicherung der Geschäftsdaten müssen diese Daten in Tabellen abgelegt werden.
- x Für die Abbildung von Objekt-Strukturen auf eine relationale Datenbank-Struktur (objektrelationales Mapping) gib es verschiedene Konzepte.
- x Einfaches Konzept:
 - ⇒ Für jede Klasse wird eine Tabelle gleichen Namens angelegt.
 - ⇒ Die Spalten der Tabelle entsprechen den Objektattributen der Klasse: gleicher Name, gleicher Typ (oder zumindest ähnlicher Typ)
 - ⇒ Die Einträge in der Tabelle (Zeilen) entsprechen den Instanzen der Klasse.

... *Objektrelationales Mapping*

- x Bei dieser Vorgehensweise können den elementaren Java-Operationen auf Objekten entsprechende SQL-Operationen gegenübergestellt werden.
 - ⇒ Der Java-Operation *new* entspricht ein *insert* in SQL.
 - ⇒ Der Wertzuweisung zu einem Objektattribut entspricht ein *update* in SQL.
- x In systematischer Weise muss bei dieser Vorgehensweise eine Zuordnung zwischen den Java-Datentypen und den SQL-Datentypen gefunden werden. Detailfragen, zu denen unterschiedliche Lösungsansätze existieren:
 - ⇒ Durch welche SQL-Datentypen werden skalare Java-Datentypen repräsentiert?
 - ⇒ Wie wird mit Objektreferenzen umgegangen?

Beispiel

Java-Klasse

```
public class Student {  
    public String nachname;  
    public String vorname;  
    public int matrikelnummer;  
}
```

Speicherung der Instanzen in einer Tabelle

<i>vorname</i>	<i>nachname</i>	<i>matrikelnummer</i>
Frank	Schmid	134141
Oliver	Singer	141354
Michael	Brandt	223116

... Beispiel

Erzeugen der Tabelle

```
CREATE TABLE `student` (  
    `vorname` VARCHAR( 20 ) ,  
    `nachname` VARCHAR( 20 ) ,  
    `matrikelnummer` INT  
);
```

Erzeugen eines Java-Objekts - Einfügeoperation in SQL

```
Student s = new Student();  
s.vorname = "Frank";  
s.nachname = "Schmid";  
s.matrikelnummer = 134141;
```

```
INSERT INTO `student`  
( `vorname`, `nachname`, `matrikelnummer` )  
VALUES ('Frank', 'Schmid', '134141');
```

4.5 Transaktionen

Transaktionen

Die Datenbank sorgt immer dafür, dass einzelne SQL-Anweisungen vollständig oder gar nicht ausgeführt werden (Atomizität).

Mehrere aufeinander folgende SQL-Anweisungen können zu einer Transaktion zusammengefasst werden. Die Datenbank sorgt dann dafür, dass die Transaktion nur als ganzes ausgeführt wird. Es werden also entweder alle ihre SQL-Anweisungen ausgeführt, oder es wird - im Falle des Scheiterns - keine der SQL-Anweisungen ausgeführt.

Auto-Commit - Commit/Rollback

- x Wenn nicht explizit anders vorgegeben, so befindet sich eine JDBC-Connection im Auto-Commit-Modus:
 - ⇒ Jede SQL-Anweisungen, die über sie verschickt wird, wird einzeln von der Datenbank ausgeführt.
 - ⇒ Im Auto-Commit-Modus ist es nicht möglich mehrere SQL-Anweisungen zu einer Transaktion zusammenzufassen.

- x Ausschalten des Auto-Commit-Modus:

```
connection.setAutoCommit(false);
```

- x Einschalten des Auto-Commit-Modus:

```
connection.setAutoCommit(true);
```

Commit

Sollen mehrere SQL-Anweisungen zu einer Transaktion zusammengefasst werden, so muss wie folgt vorgegangen werden:

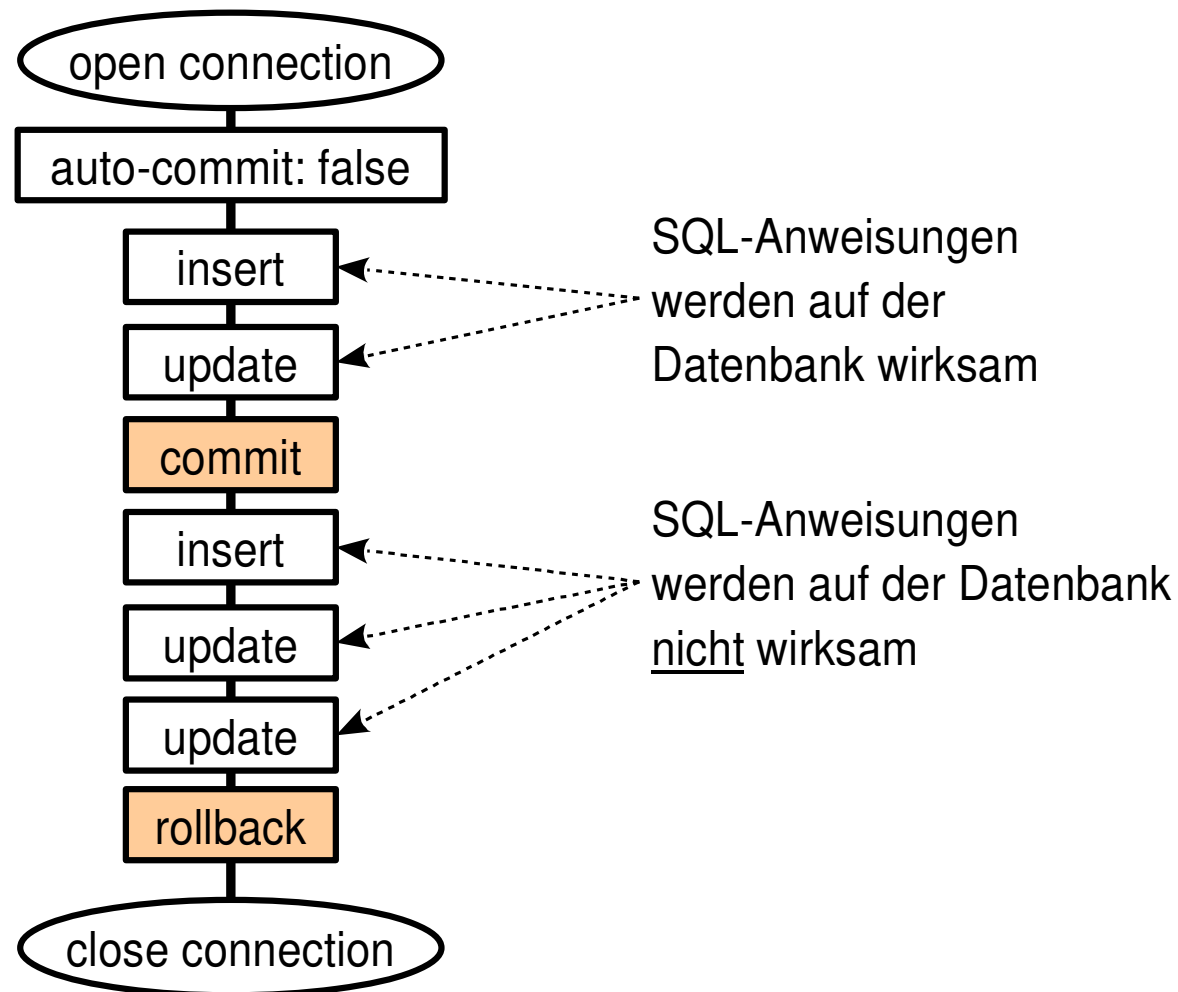
- ⇒ Man benötigt eine JDBC-Verbindung und muss dafür sorgen, dass der Auto-Commit-Modus ausgeschaltet ist.
- ⇒ Man schickt über diese Verbindung mehrere SQL-Anweisungen.
- ⇒ Man führt anschließend eine Commit-Operation durch.

```
connection.setAutoCommit(false);  
Statement statement = connection.createStatement();  
statement.executeUpdate("INSERT INTO `zahlen` (`zahl`) VALUES ('12');");  
statement.executeUpdate("INSERT INTO `zahlen` (`zahl`) VALUES ('17');");  
connection.commit();
```

Rollback

- x Mit der Rollback-Operation werden alle SQL-Anweisungen einer Connection rückgängig gemacht, die seit dem Öffnen der Verbindung oder der letzten Commit- oder Rollback-Operation ausgeführt wurden.
- x Durch Commit- und Rollback-Aufrufe werden die SQL-Anweisungen einer Connection in mehrere Abschnitte unterteilt, die Transaktionen entsprechen.
 - ⇒ Folgt dem Abschnitt eine Commit-Anweisung oder wird die Datenbankverbindung anschließend geschlossen, so werden die SQL-Anweisungen des Abschnitts auf der Datenbank wirksam.
 - ⇒ Folgt dem Abschnitt eine Rollback-Anweisung, so werden die SQL-Anweisungen des Abschnitts rückabgewickelt und somit auf der Datenbank nicht wirksam.

Beispiel



Einsatz von Rollback

Zielsetzung:

- x Innerhalb eines Programmstücks sollen mehrere SQL-Anweisungen ausgeführt werden. Die SQL-Anweisungen des Programmstücks sollen nur alle gemeinsam oder gar nicht ausgeführt werden (Transaktion).

Es kann im Programmverlauf eine Situation entstehen, die dazu führt, dass die weiteren SQL-Anweisungen der Transaktion nicht ausgeführt und die bisherigen SQL-Anweisungen rückabgewickelt werden sollen (rollback). Situationen:

- x Das Programm erkennt aufgrund bestimmter Daten, dass die Transaktion als Ganzes nicht ausgeführt werden soll, und ruft explizit rollback auf.
 - ⇒ Beispiel: Eine Kauforder mit mehreren Einzelposten soll nur als Ganzes oder gar nicht ausgeführt werden. Einige Posten sind bereits abgebucht. Vor der Buchung des nächsten Postens erkennt das Programm, dass dieses Produkt nicht mehr vorrätig ist und ruft rollback auf.

... Einsatz von Rollback

- x Es kommt innerhalb des Programmstücks bei der Ausführung einer der SQL-Anweisungen zu einer Exception. Empfohlene Vorgehensweise: SQL-Exceptions innerhalb des Programmstücks abfangen und im catch-Block rollback aufrufen.
 - ⇒ Eine Kauforder mit mehreren Einzelposten kann nur als Ganzes oder gar nicht ausgeführt werden. Einige Posten sind bereits abgebucht. Als Konsistenzbedingung wurde für die Tabelle mit dem Lagerbestand festgelegt, dass die Anzahl der vorrätigen Produkte nie negativ sein darf. Ein in der Kauforder enthaltenes Produkt sei nicht mehr vorrätig. Das Programm prüft bei der Buchung der einzelnen Posten nicht vorab, ob die Produkte noch vorrätig sind. Es kommt während der Ausführung der SQL-Anweisung, mit der der Posten zu einer Exception.

... Einsatz von Rollback

- x Es kommt irgendwo innerhalb des Programmstücks zu einer Exception, die durch das Programmstück nicht abgefangen wird (z.B. eine RuntimeException).
 - ⇒ Beispiel: Eine `ArrayIndexOutOfBoundsException`-Exception tritt auf.
 - ⇒ Empfohlene Vorgehensweise: Das Programmstück mit einem try-catch-Block umschließen mit dem alle "internen Fehler" abgefangen werden. Im Catch-Block `rollback` aufrufen und eine Fehlermeldung ausgegeben ("interner Fehler").
 - ⇒ Auf keinen Fall sollen programminterne Fehler dazu führen, dass Transaktionen nur teilweise ausgeführt werden und so ein inkonsistenter Zustand der Datenbank entsteht!

4.6 Prepared Statements

Motivation

Die Abarbeitung einer SQL-Anweisung setzt sich aus der Sicht der Datenbank aus zwei Phasen zusammen:

- ⇒ Phase 1: SQL-Anweisung parsen und daraus ableiten, welche Datenbank-Operation ausgeführt werden soll
- ⇒ Phase 2: die Datenbankoperation ausführen

Ziel:

- ⇒ Sollen mehrmals hintereinander gleiche oder einander ähnliche SQL-Anweisungen ausgeführt werden, die sich nur in einzelnen Konstanten unterscheiden, so soll die erste Phase nur einmal ausgeführt werden. Die in der ersten Phase bestimmte Datenbankoperation soll mehrfach wiederverwendet werden.

Prepared Statements

```
PreparedStatement pstatement = connection.prepareStatement(  
    "INSERT INTO studenten (vorname, nachname, matrikelnummer)" +  
    "VALUES (?, ?, ?);" );
```

```
pstatement.setString(1, "Helmut");  
pstatement.setString(2, "Müller");  
pstatement.setInt(3, 132);  
pstatement.execute();
```

```
pstatement.setString(1, "Sonja");  
pstatement.setString(2, "Klein");  
pstatement.setInt(3, 532);  
pstatement.execute();
```

Prepared Statements

- x Durch den Aufruf von `connection.prepareStatement(SQL-Anweisung)` erhält man ein `PreparedStatement`-Objekt. Das `PreparedStatement`-Objekt kann nun mehrfach hintereinander durch Aufrufen der Methode `execute` ausgeführt werden.
- x Das Parsen der SQL-Anweisung findet während der Ausführung der Anweisung `prepareStatement` statt. Das Ergebnis ist eine Datenbankoperation, auf die ab sofort über das Objekt vom Typ `PreparedStatement` zugegriffen werden kann. In der Objektmethode `execute` des `PreparedStatement` wird lediglich die Datenbankoperation ausgeführt - es findet kein Parse-Vorgang statt.
- x Wie die Datenstruktur aussieht, die die Datenbankoperation beschreibt, hängt von der Datenbank und dem Treiber ab. Es ist insbesondere auch möglich, dass diese Information serverseitig gespeichert wird, was zu einer höheren Performance führt.

Parameter in Prepared Statements

- x Fragezeichen innerhalb der SQL-Anweisungen stellen bei Prepared Statements Platzhalter dar, die von Aufruf zu Aufruf unterschiedlich belegt werden können.

```
INSERT INTO studenten (vorname, nachname, matrikelnummer)  
VALUES (?, ?, ?);
```

- x Jedes Platzhalter erhalten in der Reihenfolge des Auftretens in der SQL-Anweisung intern eine fortlaufende Nummer beginnend mit 1. In obigem Beispiel sind dies die Nummern 1, 2 und 3.

... Parameter in Prepared Statements

- x Vor der Ausführung der execute-Methode des Prepared Statement müssen alle Platzhalter mit konkreten Werten belegt werden. Dazu stehen mehrere set-Methoden zur Verfügung, die dem Platzhalter mit einer vorgegebenen Nummer einen bestimmten Wert zuordnen. Die set-Methoden gibt es für unterschiedliche Datentypen: setInt, setString etc.

```
pstatement.setString(1, "Helmut");  
pstatement.setString(2, "Müller");  
pstatement.setInt(3, 132);  
pstatement.execute();
```

Prepared Statements und Sicherheit

- x Neben Performance-Überlegungen sprechen auch Sicherheitsaspekte für Prepared Statements. Oft müssen Daten in SQL-Anweisungen eingefügt werden, die zuvor vom Benutzer eingegeben wurden.
 - ⇒ Beispiel: In der Anwendung soll der Benutzer nach Einträgen in der Tabelle suchen können, bei denen ein Attribut einen bestimmten Wert annimmt.

```
SELECT * FROM `studenten` WHERE  
    vorname = 'Eingabe';
```

- x Problem: Vom Benutzer können beliebige Daten eingegeben werden. Werden diese unverändert in die SQL-Anweisung übernommen, so kann über geschickte Eingaben die Struktur der Anweisung manipuliert werden.

```
SELECT * FROM `studenten` WHERE  
    vorname = 'Karl'; drop studenten; %';
```

...Prepared Statements und Sicherheit

- x Einbinden von Benutzerzeingaben (Variable: eingabe) in ein einfaches SQL-Statement:

```
Statement statement = connection.createStatement();  
ResultSet resultset = statement.executeQuery(  
    "SELECT * FROM studenten WHERE vorname = " + eingabe + ";" );
```

- x Einbinden von Benutereingaben in ein Prepared Statement:

```
PreparedStatement pstatement = connection.prepareStatement(  
    "SELECT * FROM studenten WHERE vorname = ?;" );  
pstatement.setString(1, eingabe);  
ResultSet resultset = pstatement.executeQuery();
```

- x Unterschied: Die Variante mit dem Prepared-Statement ist Manipulations-sicher. Bei der Statement-Variante muss vorab die eingegebene Zeichenkette validiert werden, um Manipulationen auszuschließen.

4.7 Nebenläufiger Datenbankzugriff

Connections und Performance

Alle SQL-Anweisungen, die innerhalb einer Datenbankverbindung an die Datenbank geschickt werden, werden sequentiell abgearbeitet. Bei der Bearbeitung von SQL-Anweisungen treten Leerlaufzeiten auf - so etwa beim Warten auf Daten, die von einem persistenten Datenspeicher (z.B. Festplatte) angefordert wurden. Würden alle Operationen über eine einzige Datenbankverbindung sequentiell zur Datenbank geschickt werden, so wäre die Datenbank nicht ausgelastet. Dies gilt insbesondere bei Datenbankinstallationen, die inhärent nebenläufig arbeiten:

- ⇒ mehrere Threads
- ⇒ mehrere gleichzeitig genutzte persistente Platten
- ⇒ Cluster aus mehreren Servern

Durch mehrere gleichzeitige Verbindungen zu einer Datenbank kann der Durchsatz signifikant erhöht werden.

Datenbankverbindungen und Multithreading

Empfohlene Vorgehensweise bei Java-Anwendungen mit mehreren Threads:

- ⇒ Jeder Thread greift über eine eigene Datenbankverbindung auf die Datenbank zu.

Gründe:

- ⇒ Der Durchsatz ist höher als bei einer gemeinsam genutzten Datenbankverbindung.
- ⇒ Sollen in dem Programmcode, den ein Thread abarbeitet, mehrere SQL-Anweisungen zu einer Transaktion zusammengefasst werden, so ist dies nur über eine eigene Datenbankverbindung möglich.
- ⇒ Prepared Statements, die für eine Datenbankverbindung erzeugt wurden, können nur innerhalb dieser Datenbankverbindung wiederverwendet werden.

Datenbankverbindungen aufbauen und schließen

Programme, die Datenbankverbindungen mit `datasource.getConnection()` öffnen, müssen nach der Benutzung der Datenbankverbindung dafür sorgen, dass sie wieder geschlossen wird (`close`)!

⇒ Problem:

Die Anzahl der Datenbankverbindungen, die gleichzeitig geöffnet werden können, ist begrenzt. Durch geöffnete und nicht wieder geschlossene Verbindungen werden diese Ressourcen aufgebraucht.

Datenbankverbindungen und Exceptions

Das Schließen der Datenbankverbindung muss insbesondere auch dann gewährleistet sein, wenn es innerhalb eines Programmstücks zu einer Exception kommt. Empfehlung: Try-With-Resources-Block

```
try (Connection connection = datasource.getConnection();) {  
    Statement statement = connection.createStatement();  
    ResultSet resultSet =  
        statement.executeQuery("select * from studenten");  
    ...  
}
```

- x Das Anfordern der Verbindung erfolgt direkt nach dem Schlüsselwort try in dem Abschnitt mit den runden Klammern:
 - ⇒ Deklaration der Variablen und Anforderung der Ressource
- x Der zugehörige close-Befehl, mit dem die Connection geschlossen wird, wird automatisch aufgerufen.

... Datenbankverbindungen und Exceptions

- x Der Try-With-Resources-Block kann nur mit Klassen verwendet, die von der Schnittstelle *AutoCloseable* erben und deshalb eine *close*-Methode haben.
- x Diese *close*-Methode wird immer automatisch aufgerufen, wenn der Rumpfblock des try-Blocks verlassen wird. Und zwar ganz egal, wie dieser Block verlassen wird. Möglichkeiten den try-Block zu verlassen:
 - ⇒ regulärer Durchlauf ohne Exception
 - ⇒ es kommt zu einer Exception, die in diesem try-Block mit einem catch abgefangen wird
 - ⇒ es kommt zu einer Exception, die in diesem try-Block nicht mit einem catch abgefangen wird, vielleicht wird sie in einem anderen catch-Block außerhalb abgefangen, vielleicht gar nicht

Datenbankverbindungen und Web-Applications

- x Durch den Application Server werden die service-Methoden der Servlets von mehreren Working-Threads nebenläufig, in reentranter Weise ausgeführt.
- x Empfohlene Vorgehensweise:
 - ⇒ Für jeden Aufruf der service-Methode eine Datenbankverbindung.
 - ⇒ Wird innerhalb der service-Methode eine Datenbankverbindung benötigt, so wird die Datenbankverbindung zu Beginn der service-Methode hergestellt und am Ende der Service-Methode wieder geschlossen.

4.8 Connection Pooling

Motivation

x Gegeben:

- ⇒ Ein nebenläufiger Service, der mit Hilfe eines Dispatching-Threads und mehreren Working-Threads realisiert wurde.
- ⇒ Die Working-Threads erbringen die Dienstleistungen und benötigen dazu Zugriff auf eine Datenbank.
- ⇒ Wann immer die Working-Threads vom Dispatcher mit der Bearbeitung einer Aufgabe betraut werden, bei denen sie auf die Datenbank zugreifen müssen, öffnen sie zu Anfang eine Datenbankverbindung und schließen sie anschließend wieder.

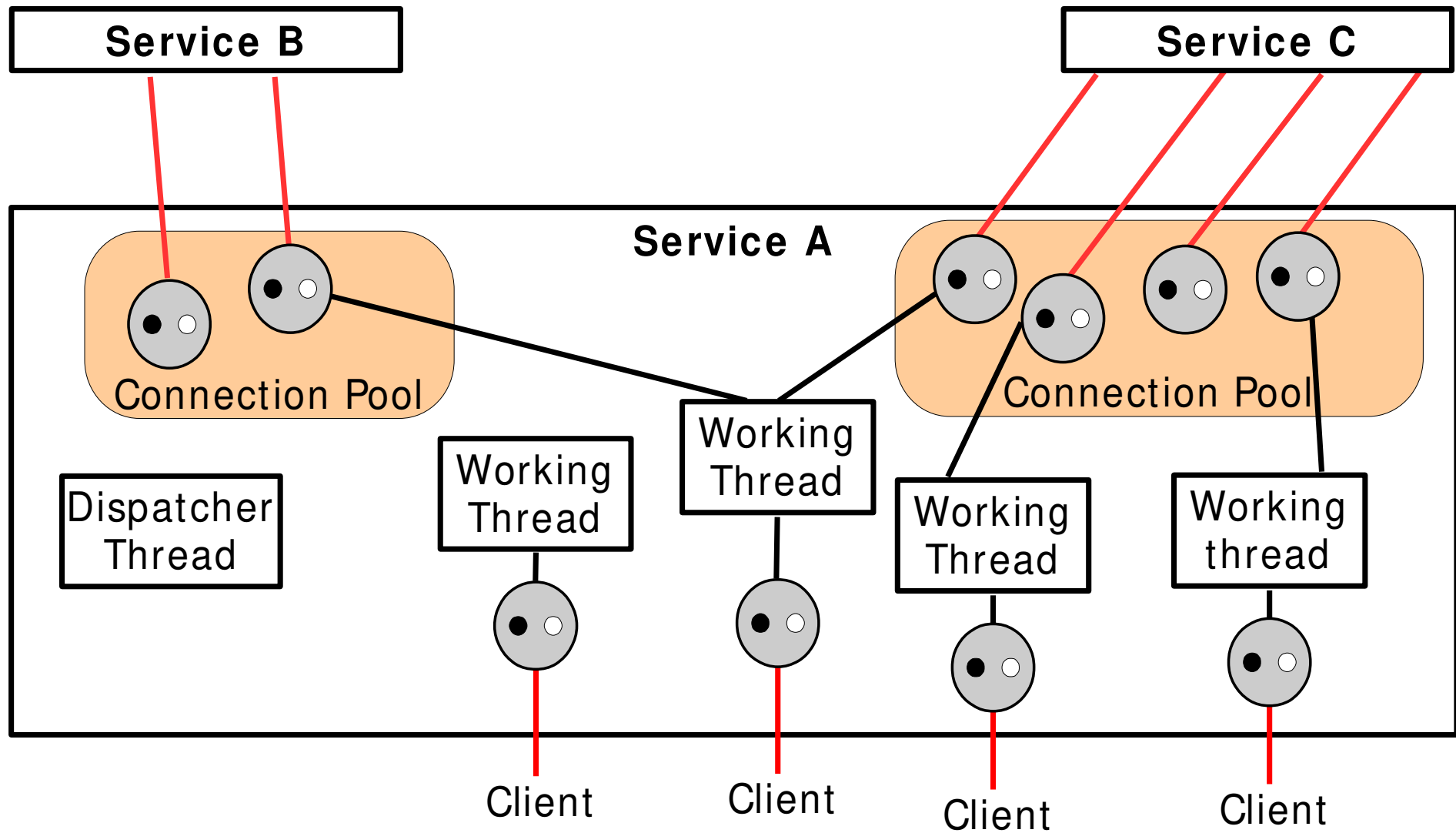
x Problem:

- ⇒ Das häufige Öffnen und Schließen einer Datenbankverbindung ist aufwändig: Herstellen einer Netzwerk-Verbindung für den Fall, dass die Datenbank nicht auf dem gleichen Computer installiert ist. Authentifizierung. Abbau der Verbindung.

Idee

- x Zur Optimierung des Datenbankzugriffs kann die DataSource so implementiert werden, dass eine reale Datenbankverbindung mehreren Nutzern, die eine Datenbankverbindung benötigen, nacheinander zugeteilt wird.
- x Man verwendet hierzu ein Konzept, das allgemein als Connection Pooling bezeichnet wird.
- x Voraussetzungen für Connection Pooling:
 - ⇒ Eine Anwendung ist intern nebenläufig mit mehreren Threads realisiert.
 - ⇒ Die Anwendung, d.h. die verschiedenen Threads, wollen als Client auf einen Service zugreifen.
 - ⇒ Der Service unterstützt mehrere gleichzeitige Verbindungen.

Connection Pooling



Konzept Connection Pooling

- x Offene Verbindungen zu dem Service werden in einem Connection Pool (einer Datenstruktur) aufbewahrt und verwaltet.
- x Der Connection Pool stellt zwei Methoden zur Verfügung über die man die Verbindung temporär für sich reservieren und wieder freigeben kann: *getConnection*, *close*
- x Wird mit *getConnection()* eine Verbindung angefordert, so wird dem Aufrufer eine gerade ungenutzte Verbindung aus dem Connection Pool zugeteilt.
- x Durch den *close()*-Aufruf wird die Datenbankverbindung nicht abgebaut, sondern die Datenbankverbindung wird lediglich in den Pool der gerade ungenutzten Datenbankverbindungen zurückgegeben.

Realisierung in JDBC

- x *DataSource* ist Bestandteil der JDBC-API. Es gibt Implementierungen von *DataSource*, die intern Connection-Pooling betreiben und andere, die ohne Connection-Pooling direkte Verbindungen zur Datenbank aufbauen.
- x Aus der Sicht der Anwendung ergibt sich kein Unterschied. Die Anwendung arbeitet in beiden Fällen mit einem *DataSource*-Objekt, über das sie mit *getConnection()* eine Datenbankverbindung erhält und die sie dann mit *close()* wieder freigibt.
- x Ob die *DataSource* Connection-Pooling betreibt oder ob mit den Operationen *getConnection()* und *close()* tatsächlich immer eine neue Verbindung zur Datenbank auf- bzw. abgebaut wird, ist aus Sicht der Anwendung nicht erkennbar. Eine einmal erstellte Anwendung funktioniert in beiden Betriebsmodi. Durch Konfiguration kann flexibel zwischen Datenbanktreibern mit und ohne Connection-Pooling gewechselt werden. Im allgemeinen führen Datenbanktreiber, die Connection-Pooling unterstützen, zu einer deutlich höheren Performance der Anwendung.

Datenbankverbindungen schließen!

- x Unabhängig davon, ob eine Anwendung mit oder ohne Connection-Pooling betrieben wird, muss von der Anwendung immer gewährleistet werden, dass die von ihr angeforderten Verbindungen auch wieder geschlossen werden.
- x Unterschiedliche Auswirkung, wenn Datenbankverbindungen nicht geschlossen werden:
 - ⇒ Wird die Anwendung ohne Connection-Pooling betrieben, so kann es dazu führen, dass zu viele Verbindungen zur Datenbank aufgebaut werden. Die Anzahl der Verbindungen zu einer Datenbank kann seitens der Datenbank limitiert werden.
 - ⇒ Wird Connection-Pooling eingesetzt, so wird im Connection-Pool eine Obergrenze für die realen Datenbankverbindungen definiert, die gleichzeitig zur Datenbank aufgebaut werden. Datenbankverbindungen, die angefordert und nicht wieder geschlossen wurden, können nicht wieder zugeteilt werden.

Pooling von Prepared Statements

```
PreparedStatement pstatement = connection.prepareStatement(  
    "SELECT * FROM studenten WHERE vorname = ?;" );
```

- x Ab JDBC 3.0 werden *PreparedStatement*-Objekte in einem Pool gepuffert.
- x Wird in einer Datenbankverbindung zum ersten Mal die Methode *prepareStatement* mit einer bestimmten SQL-Anweisung aufgerufen, so wird der SQL-String in eine Datenstruktur überführt, die die Datenbankoperation repräsentiert. Die Datenstruktur wird in einem Pufferspeicher, dem Statement-Pool abgelegt.
- x Wird in der gleichen Verbindung die Methode *prepareStatement* wieder mit der gleichen SQL-Anweisung aufgerufen, so muss die Datenstruktur, die die Datenbankoperation repräsentiert, nicht erneut erstellt werden, sondern sie wird einfach aus dem Statement-Pool gelesen.

... Pooling von Prepared Statements

- x Der Pufferungsmechanismus bezieht sich auf reale Datenbankverbindungen. Wurde das *PreparedStatement* in einer Datenbankverbindung berechnet, und wird diese Datenbankverbindung im Zuge des Connection-Pooling an einen anderen Nutzer weitergereicht, so steht das *PreparedStatement*-Objekt dort bereits im Statement-Pool zur Verfügung.

Empfohlene Vorgehensweise

- x Soll eine SQL-Anweisung mit einem *PreparedStatement* ausgeführt werden, so ruft man zuvor immer die *prepareStatement*-Methode auf.
- x Das *PreparedStatement*-Objekt muss nicht in irgend einer Variablen gespeichert werden, damit es beim nächsten *execute*-Aufruf wiederverwendet werden kann. Wurde die SQL-Anweisung innerhalb der Datenbankverbindung schon einmal in ein *PreparedStatement* umgewandelt, so wird das Objekt durch diesen Aufruf nicht erneut berechnet, sondern es wird einfach das zuletzt berechnete Objekt aus dem Statement-Pool gelesen.

4.9 Transaction Isolation

Transaction Isolation

- x Bei Datenbanktransaktionen, die sich aus mehreren SQL-Anweisungen zusammensetzen, ist immer gewährleistet, dass sie von der Datenbank ganz oder gar nicht ausgeführt werden. Datenbanken arbeiten nebenläufig. Während eine Transaktion ausgeführt wird, können zeitgleich andere SQL-Anweisungen ausgeführt werden, die Daten ändern, auf die in der Transaktion zugegriffen wird. Auch durch die Transaktion selbst können diese Daten verändert werden.
- x Offene Frage: Wie wirken sich diese Änderungen auf die Transaktion aus?
- x Dies ist in unterschiedlichen Datenbankimplementierungen unterschiedliche gelöst und es kann oft auch zwischen unterschiedlichen Modi hin- und hergeschaltet werden.
- x In JDBC werden diese Modi als "Transaction Isolation Levels" bezeichnet.
 - ⇒ Wie wird die Transaktion von den Auswirkungen anderer Transaktionen abgeschirmt?

Transaction Isolation Levels

- x JDBC unterscheidet fünf Transaction Isolation Levels, die durch statische final-Attribute (Konstante) der Klasse Connection repräsentiert werden:

- ⇒ TRANSACTION_NONE
- ⇒ TRANSACTION_READ_UNCOMMITTED
- ⇒ TRANSACTION_READ_COMMITTED
- ⇒ TRANSACTION_REPEATABLE_READ
- ⇒ TRANSACTION_SERIALIZABLE

- x Man kann von einer Datenbankverbindung erfragen, welche der Modi sie unterstützt und man kann in einen unterstützten Modus wechseln.

```
boolean b = connection.getMetaData().supportsTransactionIsolationLevel(
    Connection.TRANSACTION_READ_COMMITTED);
connection.setTransactionIsolationLevel(
    Connection.TRANSACTION_READ_UNCOMMITTED);
```

... Transaction Isolation Levels

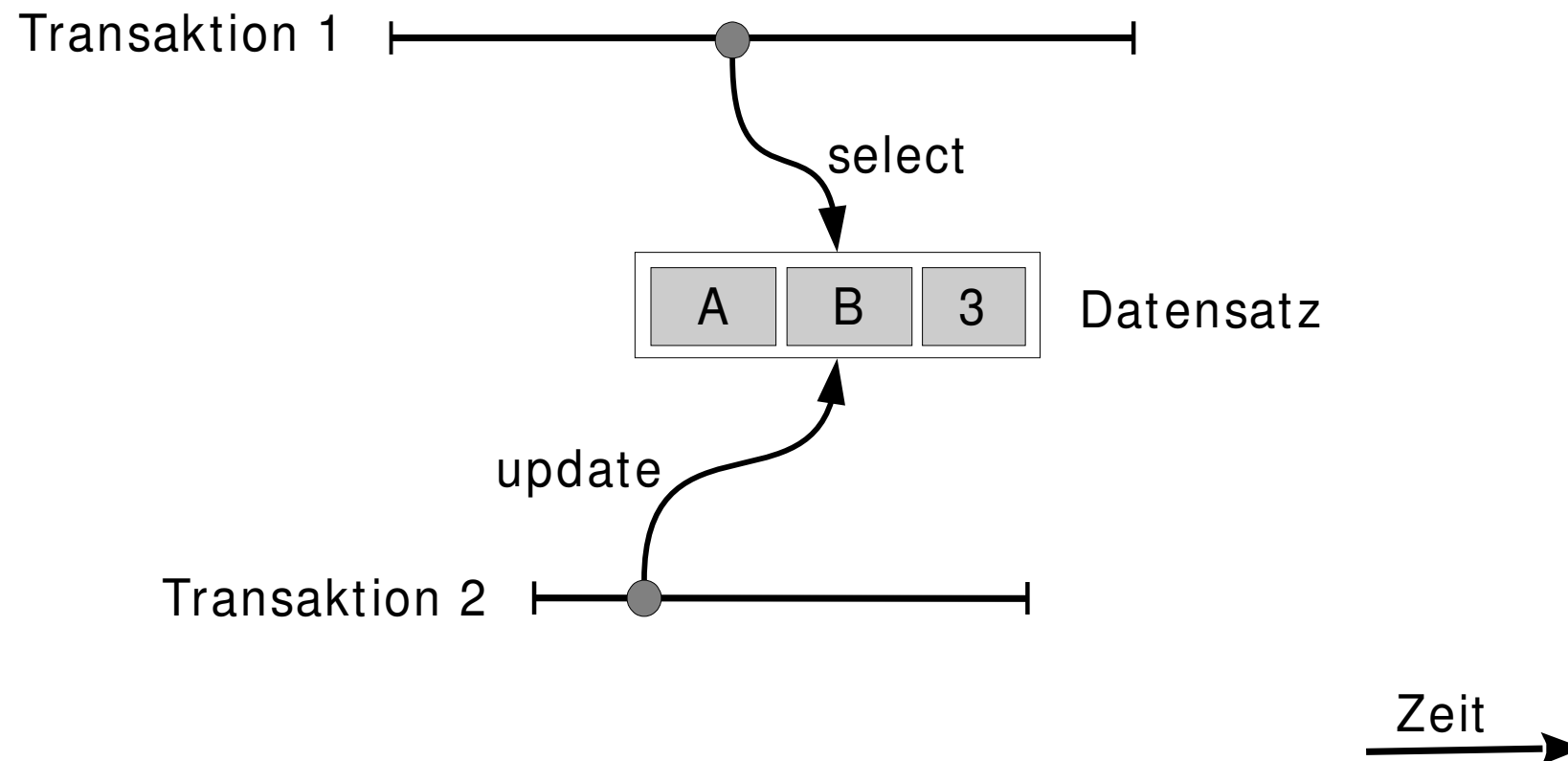
- x Unterstützt die Datenbankverbindung keine Transaktionen, so ist TRANSACTION_NONE die einzige unterstützte Transaction Isolation Level. Nach jeder einzelnen SQL-Anweisung wird implizit eine Commit-Operation ausgeführt.
- x Unterstützt die Datenbankverbindung Transaktionen, so unterstützt sie eine nichtleere Teilmenge der anderen vier Transaction Isolation Levels TRANSACTION_READ_UNCOMMITTED, TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ und TRANSACTION_SERIALIZABLE. Der Modus TRANSACTION_NONE wird dann nicht unterstützt.

Dirty Reads

Darf in einer Transaktion auf Daten zugegriffen werden, die durch die Ausführung einer Transaktionen entstanden sind, die zu diesem Zeitpunkt noch nicht abgeschlossen (committed) ist?

- ⇒ Diese Lese-Vorgänge bezeichnet man als "Dirty Reads".
- ⇒ Problem: Es könnte sein, dass die noch nicht abgeschlossene Transaktion nachfolgend mit rollback rückabgewickelt wird und so diese Daten keinen gültigen Zustand der Datenbank repräsentieren.
- ⇒ Um Dirty Reads zu vermeiden, dürfen für Datenbanktransaktion immer nur die Datenänderungen sichtbar sein, die sich aus abgeschlossenen Transaktionen ergeben. Zwischenzustände einer Transaktion dürfen nach außen nicht sichtbar sein.

... Dirty Reads

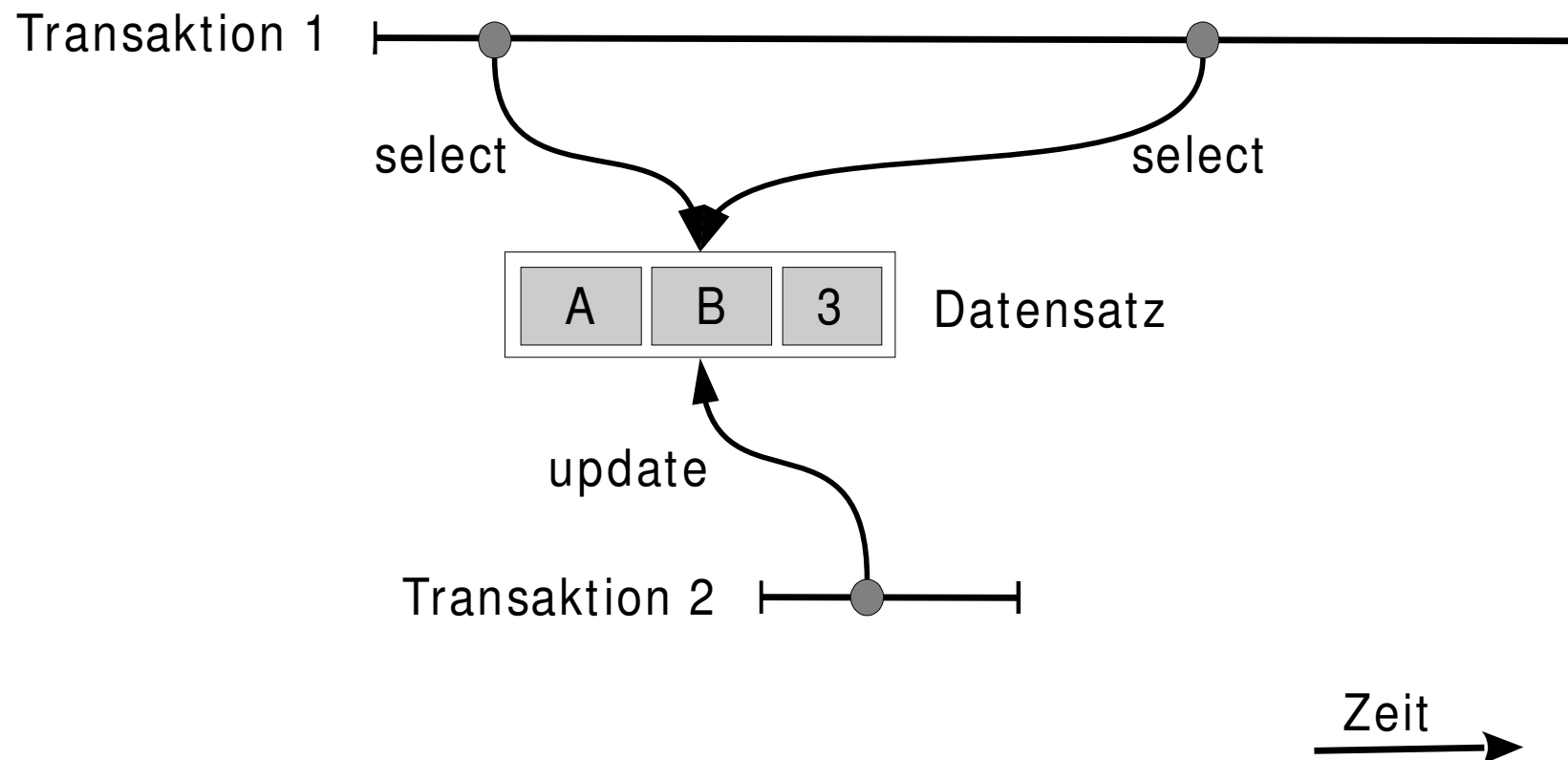


Nonrepeatable Reads

Darf es passieren, dass in einer Transaktion zwei nacheinander ausgeführte Lesevorgänge auf der selben Zeile zu unterschiedlichen Ergebnissen führen, da die Zeile zwischen den beiden Lese-Vorgängen von einer Transaktion geändert wurde?

- ⇒ Man bezeichnet dieses Phänomen als "Nonrepeatable Reads".
- ⇒ Um Nonrepeatable Reads zu vermeiden, muss die Datenbank die Zeile während der Ausführung der Transaktion für Schreibzugriffe sperren.

... Nonrepeatable Read

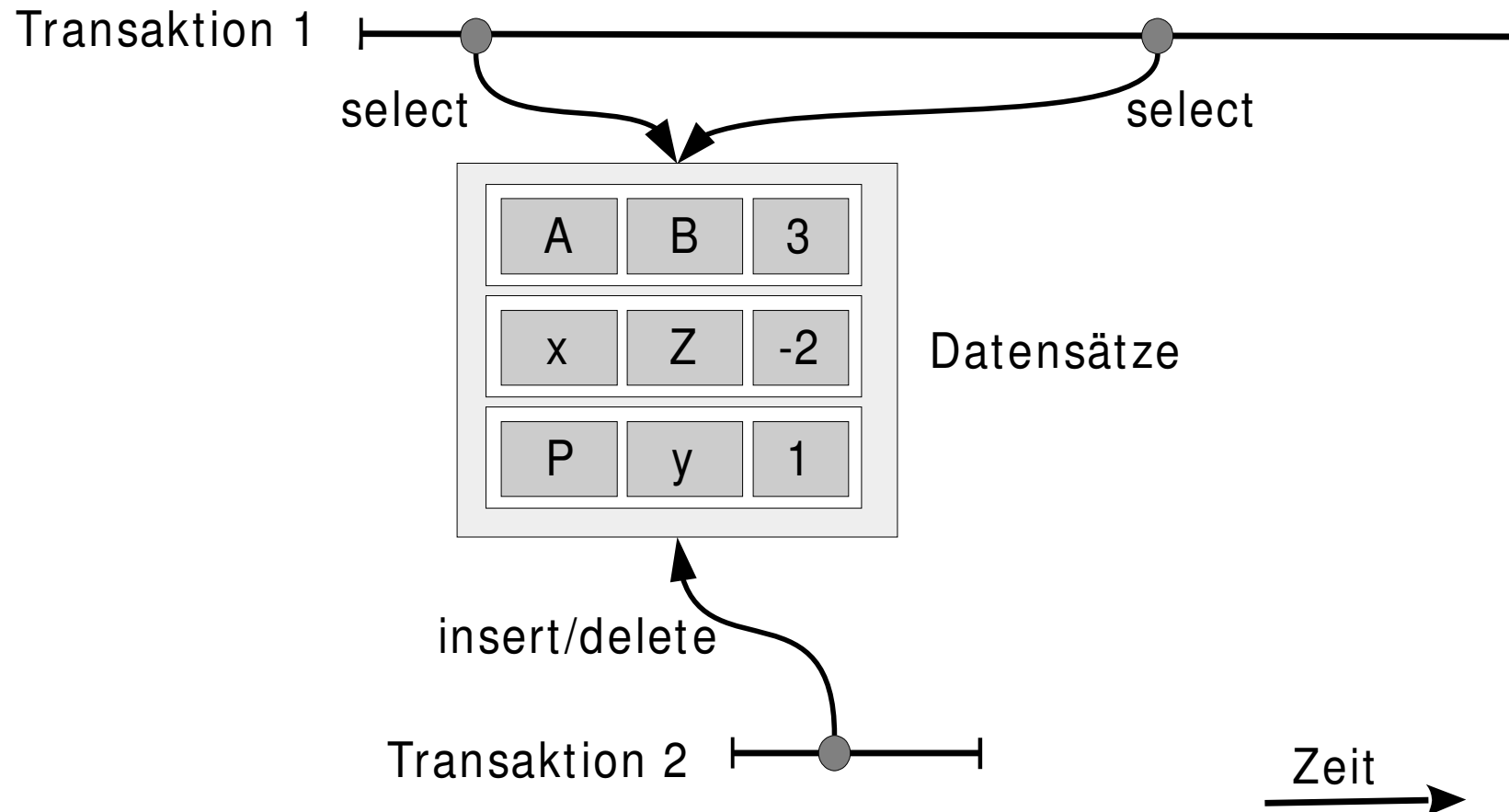


Phantom Reads

In einer Transaktion werden hintereinander zwei gleiche Lesevorgänge durchgeführt, die jeweils mit einer WHERE-Klausel eine Menge von Zeilen auswählen. Darf es passieren, dass durch sie das Ergebnis beim zweiten Lesevorgang um eine Zeile vergrößert, da zwischenzeitlich durch eine andere Transaktion eine Zeile eingefügt wurde, die die WHERE-Klausel erfüllt?

- ⇒ Man bezeichnet dieses Phänomen als "Phantom Reads".
- ⇒ Um Phantom Reads zu vermeiden, muss die Datenbank alle Zeilen der Tabelle für Einfügeoperationen sperren, die die WHERE-Klausel erfüllen.

... Phantom Read



Transaction Isolation Levels

	<i>dirty reads</i>	<i>non-repeatable reads</i>	<i>phantom reads</i>
TRANSACTION_READ_UNCOMMITTED	möglich	möglich	möglich
TRANSACTION_READ_COMMITTED	ausgeschlossen	möglich	möglich
TRANSACTION_REPEATABLE_READ	ausgeschlossen	ausgeschlossen	möglich
TRANSACTION_SERIALIZABLE	ausgeschlossen	ausgeschlossen	ausgeschlossen

Durchsatz

Die Entscheidung für eine bestimmte Transaction Isolation Level bestimmt, welcher Aufwand bei der Durchführung von Transaktionen für Sperren aufgewandt werden muss und hat somit maßgeblichen Einfluss auf den mit der Datenbank erreichbaren Durchsatz an Transaktionen (Performance).

	<i>Durchführung von Transaktionen</i>	<i>Durchsatz der Datenbank</i>
TRANSACTION_NONE	kein Aufwand für Sperren : : hoher Aufwand für Sperren	hoch : gering
TRANSACTION_READ_UNCOMMITTED		
TRANSACTION_READ_COMMITTED		
TRANSACTION_REPEATABLE_READ		
TRANSACTION_SERIALIZABLE		

Kapitel 5

Web-Applications

HTTP/HTML

- x Den Durchbruch auf den Massenmarkt hat das Internet ohne Zweifel den beiden Standards HTTP und HTML zu verdanken.
- x Browser auf der Basis von HTTP/HTML stellen eine einfach zu bedienende und universell einsetzbare Schnittstelle zum Benutzer dar.
 - ⇒ Prinzip: Der Browser holt via HTTP-Protokoll eine Web-Seite vom Server und stellt den mit HTML kodierten Inhalt auf dem Bildschirm dar. Mausklicks auf Links in den HTML-Seiten führen zu weiteren Web-Seiten.
- x Dieses einfache Konzept hat auch heute noch eine dominante Stellung im Internet.
- x HTTP und HTML sowie zahlreiche andere weit verbreitete "Internet-Standards" werden durch das W3C (world wide web consortium) formal spezifiziert und weiterentwickelt (<http://www.w3.org>).

5.1 Das Protokoll HTTP

HTTP und HTTPS

- x Das Anwendungsprotokoll HTTP dient dem Austausch von Daten zwischen Client und Server.
- x Browser sind HTTP-Clients.
- x HTTP-Server bezeichnet man gerne als Web-Application-Server.
- x HTTPS: eine mit SSL gesicherte Variante von HTTP
 - ⇒ HTTP over SSL
- x Wenn im Folgenden die Funktionsweise des HTTP-Protokolls beschrieben wird, so sind diese Aussagen in gleicher Weise auch auf HTTPS anwendbar. In HTTP und HTTPS finden die selbe Form der Kommunikation statt – einmal unverschlüsselt und einmal verschlüsselt.

Request-Response-Prinzip

- x Das HTTP-Protokoll folgt dem Request-Response-Prinzip: Der Service antwortet auf jeden HTTP-Request des Clients mit einer HTTP-Response. Der Aufbau des HTTP-Requests und der HTTP-Response wird durch das HTTP-Protokoll festgelegt.
- x Einen Request-Response-Vorgang bezeichnet man bei HTTP zusammenfassend als eine HTTP-Message.
- x Über eine TCP/IP-Verbindung können mehrere HTTP-Messages (Request-Response-Aktionen) hintereinander ausgeführt werden. Es kann jedoch auch für jede HTTP-Message eine eigene Verbindung hergestellt werden.²

² Dies gilt seit HTTP 1.1. Bei älteren HTTP-Versionen musste für jede HTTP-Message ein TCP/IP-Verbindung aufgebaut werden.

HTTP-URL, HTTP-Methode, Dateneinheiten

- x Das HTTP-Protokoll arbeitet mit Dateneinheiten, die von Servern zur Verfügung gestellt werden und denen jeweils eine weltweit eindeutige Adresse (HTTP-URL) zugeordnet ist.
- x Das HTTP-Protokoll beschreibt Operationen, mit denen man auf diese Daten zugreifen und sie verändern kann. Diese Operationen bezeichnet man als HTTP-Methoden. In jedem Request wird angegeben, mit welcher Dateneinheit welche Operation durchgeführt werden soll.

HTTP-URLs und andere URLs

- x URL steht für Uniform Resource Locator
- x Ein URL ist eine einzeilige Zeichenkette mit einem standardisierten Aufbau.
- x URLs gibt es nicht nur im Bereich HTTP/HTTPS

- x Aufbau eines URL:

https	://	www.bundestag.de	:	443	/	press
<i>Protokoll</i>		<i>Hostname / IP-Adresse</i>		<i>Port</i>		<i>Pfad</i>

- x Der Doppelpunkt und die Portnummer dürfen weggelassen werden, wenn die Portnummer die Default-Portnummer des Anwendungsprotokolls ist. Bei HTTPS ist die Defaultportnummer 443. Deshalb darf man diesen URL auch verkürzt schreiben als
https://www.bundestag.de/press

... HTTP, HTTP-URL

- x Die Spezifikation von HTTP macht bezüglich der Daten, die einem HTTP-URL zugeordnet sind, keine Einschränkungen. Der Inhalt und das Format der Daten (Kodierung) können prinzipiell beliebig sein.
- x Die einem URL zugeordneten Daten werden gerne auch als Seiten bezeichnet. Im engeren Sinne bezieht sich diese Bezeichnung auf Inhalte im HTML-Format. Der Begriff Seite wird jedoch gerne auch für Daten in anderen Formaten verwendet.
- x Im einfachsten Fall sind die Daten, die einem URL zugeordnet sind, nichts anderes als der Inhalt einer Datei auf dem Server, die der Server an dem im HTTP-URL angegebenen Dateipfad in seinem Dateisystem abgelegt hat. Dies muss jedoch nicht so sein. Welche Daten einer bestimmten HTTP-URL zugeordnet sind, kann ein HTTP-Service in beliebiger Weise festlegen.

HTTP-URL, HTTP-Methode

Ein HTTP-Request enthält neben der HTTP-URL auch eine HTTP-Methode. Die HTTP-Methode beschreibt, was mit den Daten der URL gemacht werden soll. Es gibt acht verschiedene HTTP-Methoden:

GET	Daten vom Server holen
POST	Daten zum Server übertragen
HEAD	nur den Header der Response anfordern
PUT	Daten auf dem Server ändern
DELETE	Daten auf dem Server löschen
OPTIONS	Kommunikationsoptionen vom Server abfragen
TRACE	Bestimmung des Übertragungsweges (Proxies)
CONNECT	Verwendung des HTTP-Protokolls für Tunnel

HTTP-Methoden

- ⇒ Die GET-Methode ist die am häufigsten gebrauchte: "Abruf von Webseiten"
- ⇒ Die POST-Methode eignet sich für die Übermittlung von Formularinhalten.
- ⇒ Mit einem HEAD-Zugriff wird nur der Header des entsprechenden GET-Zugriffs abgefragt. Ohne dass die Nutzdaten zum Client übertragen werden müssen, kann so vorab abgefragt werden, wie groß die zu übertragende Datenmenge wäre und wann die Daten zuletzt geändert wurden.
- ⇒ PUT und DELETE-Zugriffe werden in der Regel vom Server abgelehnt.
- ⇒ OPTIONS, TRACE und CONNECT zielen auf fortgeschrittene Konzepte ab.

MIME Types

- x Die einem HTTP-URL zugeordneten Daten können ein beliebiges Format haben. Bei der Übertragung dieser Daten vom Client zum Server oder vom Server zum Client kann als zusätzliche Information ein sogenannter "MIME-Type" übertragen werden, der angibt, welches Format die Daten haben.
- x MIME: multipurpose internet mail extensions
- x MIME Types sind Kurzbezeichnungen für Datenformate, die in der Regel weltweit eindeutig und standardisiert sind.
- x Beispiele:

<i>MIME Type</i>	<i>Bedeutung</i>
/text/html	formatierter Text in HTML-Codierung
/text/plain	unformatierter, reiner Text
/image/gif	Bitmap im GIF-Format

... MIME Types

- x MIME wurde ursprünglich nur für E-Mail-Inhalte definiert, sie eignen sich jedoch ganz allgemein zur expliziten Kennzeichnung des Datenformats wo immer beliebige Datenformate gespeichert oder transportiert werden sollen.
- x Ein Client, der Daten zusammen mit einem MIME-Type empfängt, wird die weitere Bearbeitung der Daten vom MIME-Type abhängig machen: Er startet eine dem MIME-Type zugeordnete Prozedur oder Anwendung.

MIME Types und Datei-Endungen

- x Werden Daten in einer Datei gespeichert, so wird die Information über das Datenformat üblicherweise in den Dateinamen als "Datei-Endung" hineinkodiert.
- x Leider ist dieses Schema nicht eindeutig und insbesondere auch nicht unabhängig vom Betriebssystem. Es werden deshalb auf den verschiedenen Betriebssystemen für die entsprechenden Werkzeuge Zuordnungen zwischen Dateiendungen und MIME-Types definiert.
- x Beispiele:

<i>MIME Type</i>	<i>Bedeutung</i>	<i>Dateiendung</i>
/text/html	formatierter Text in HTML-Codierung	.html .htm
/text/plain	unformatierter, reiner Text	.txt
/image/gif	Bitmap im GIF-Format	.gif

Aufbau eines HTTP-Requests

Erste Header-Zeile

Die erste Zeile enthält, durch Leerzeichen voneinander getrennt, die folgenden drei Informationen: die HTTP-Methode, den HTTP-URL und die HTTP-Versionsnummer

Weitere Header-Zeilen (optional)

Abhängig von der HTTP-Methode folgen dann mehrere Header-Zeilen mit technischen Detail-Informationen. Unter anderem kann dies die Länge und der MIME-Type der Nutzdaten sein, für den Fall, dass mit dem Request Nutzdaten an den Server übertragen werden sollen.

Eine Leerzeile

Nutzdaten (optional)

Ob es Nutzdaten gibt oder nicht, hängt von der HTTP-Methode ab.

Beispiel für einen HTTP-Request

```
GET /presse HTTP/1.1  
Host: www.bundestag.de
```

- x Ziel dieses Request ist es, den Inhalt des folgenden URL vom Server ausgeliefert zu bekommen:
`https://www.bundestag.de/presse`
- x Dazu wird vorab eine SSL-Verbindung mit dem Server `www.bundestag.de` aufgebaut. Anschließend wird dieser Request, der aus drei Zeilen Text besteht, über dieses Verbindung vom Client zum Server geschickt.
- x Der Request besteht aus: der ersten Header-Zeile, einer weiteren Header-Zeile und einer Leerzeile.

... Beispiel für einen HTTP-Request

- x Dieser HTTP-Request enthält keine Nutzdaten. Da liegt daran, dass es sich um einen GET-Request handelt. Bei anderen Arten von Requests kann es passieren, dass der Request nach der Leerzeile noch Nutzdaten enthält, die als Teil des Requests vom Client zum Server transportiert werden.
- x Da es sich bei diesem Request um einen HTTP-Request in der Version 1.1 handelt, ist die Header-Zeile *Host:...* obligatorisch. Hier muss der Host-Name beziehungsweise die IP-Adresse stehen

Aufbau einer HTTP-Response

Erste Header-Zeile (Statuszeile)

Die erste Zeile enthält, durch Leerzeichen voneinander getrennt, die folgenden drei Informationen: die HTTP-Versionsnummer, einen Fehlercode und eine verbale Beschreibung des Fehlercodes

Weitere Header-Zeilen (optional)

Abhängig von der HTTP-Methode folgen dann mehrere Header-Zeilen mit technischen Detail-Informationen. Werden über die HTTP-Response beispielsweise Nutzdaten übermittelt, so steht hier in der Regel unter anderem der MIME-Type (Content-Type) .

Eine Leerzeile

Nutzdaten (optional)

Ob es Nutzdaten gibt oder nicht, hängt von der HTTP-Methode ab.

Beispiel für eine HTTP-Response

```
HTTP/1.1 200 OK
Date: Fri, 21 Sep 2018 13:14:42 GMT
Server: Apache
Content-Type: text/html; charset=UTF-8
Content-Language: de
Vary: Accept-Encoding
Age: 141
X-Cache: HIT from squid:149
X-Cache-Lookup: HIT from squid:149:80
Connection: close
Access-Control-Allow-Origin: *
Cache-Control: max-age=900
```

```
<!DOCTYPE html>
<html xml:lang="de" lang="de" dir="ltr">
<head>
  <meta charset="utf-8">
...
```

... Beispiel für eine HTTP-Response

Erläuterungen:

- x Es handelt sich um das Protokoll HTTP1.1
- x Die Bearbeitung war erfolgreich: HTTP-Statuscode 200, verbale Beschreibung "OK"
- x In den nachfolgenden Zeilen des Headers folgen zahlreiche technische Details.
- x Nach der Leerzeile folgen die Nutzdaten, das heißt die dem URL zugeordneten Daten. In diesem Fall handelt es sich um eine HTML-Seite.

"HTTP ist verbindungslos"

HTTP als verbindungslos zu bezeichnen, scheint auf den ersten Blick verwirrend, da man ja eine TCP/IP-Verbindung benötigt, um HTTP-Messages (Request/Response) auszuführen.

Gemeint ist mit dieser Aussage, dass es für HTTP-Messages keinen Verbindungs-Kontext gibt, der sich aus der TCP/IP-Verbindung ergibt.

- ⇒ Jede HTTP-Message steht für sich.
- ⇒ Wie HTTP-Messages übertragen werden - über eine TCP/IP-Verbindung oder über mehrere - hat keine Bedeutung.
- ⇒ Die HTTP-Messages einer TCP/IP-Verbindung stehen in keinem besonderen Bezug zueinander.
 - x Bei anderen Anwendungsprotokollen hängen Responses nicht nur vom zuletzt verschickten Request, sondern auch von der Vorgeschichte der TCP/IP-Verbindung - den Requests, die zuvor auf der gleichen Verbindung verschickt wurden - ab. Dies ist bei HTTP nicht so.

HTTP-Sessions

- x Auf Web-Anwendungen sollen mehrere Benutzer gleichzeitig zugreifen können.
- x Bei der Konstruktion von Web-Anwendungen ist es oft notwendig, die aufeinander folgenden HTTP-Requests eines Client einander zuzuordnen.
 - ⇒ Client A klickt auf "Buch x einkaufen".
 - ⇒ Client B klickt auf "Buch y einkaufen".
 - ⇒ Client A klickt auf "CD y einkaufen".
 - ⇒ Client B klickt auf "zur Kasse".
 - ⇒ Client A klickt auf "zur Kasse".
- x Mehrere aufeinander folgende Requests des gleichen Clients bezeichnet man als eine HTTP-Session, die Verwaltung der Sessions seitens des Servers als HTTP-Session-Management.

... HTTP-Sessions

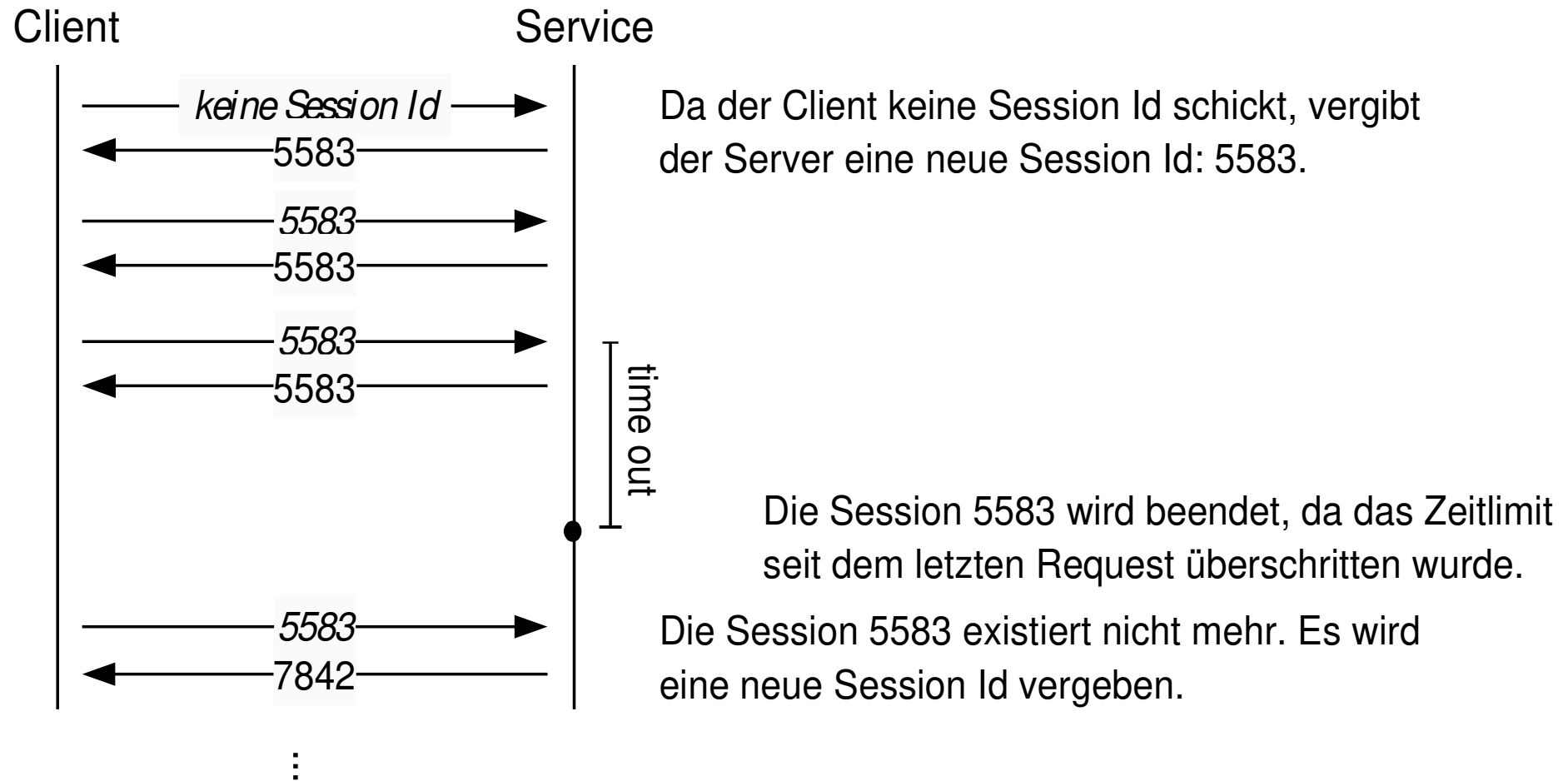
- x Eine HTTP-Session beginnt immer damit, dass ein neuer Client einen URL-Request an den Server schickt.
- x Der Server ordnet alle darauf folgenden HTTP-Requests des gleichen Clients einer Session zu.
- x Kommt von einem Client eine bestimmte Zeit lang keine HTTP-Request mehr, so wird die Session vom Server als beendet betrachtet (time out).

Session Ids

Zur eindeutigen Identifikation von Sessions vergibt der Server den Sessions Nummern, die als Session Id bezeichnet werden. Session Ids bilden die Grundlage für das Session Management:

- x Der Server beantwortet einen Request ohne Session Id damit, dass er eine neue Session-Id in der Response zurückschickt ("neuer Client").
- x Ab sofort schickt der Client diese Session-Id mit jedem weiteren HTTP-Request zu diesem Server.
- x Erhält der Server einen Request, der eine Session Id enthält, so kann er sie einer Session zuordnen. Er schickt diese Session Id auch wieder in der Response zurück.
- x Der Server verwaltet eine Liste mit allen Session-Ids. Darin wird unter anderem auch der Zeitpunkt des letzten Requests dieser Session gespeichert. Kommt es für eine definierte Zeit (time out) zu keinem Request, so wird die Session beendet (Event). Weitere Requests mit dieser Session-Id werden einer neuen Session zugeordnet.

Session Ids



Cookies, URL-Rewriting

- x Aufbauend auf dem HTTP-Protokoll gibt es zwei verbreitete Techniken für das Session-Management.
 - ⇒ Cookies
 - ⇒ URL-Rewriting
- x Sowohl Cookies als auch URL-Rewriting sind Mechanismen, mit denen Client und Server Session-Ids hin- und herschicken können. Sie unterscheiden sich lediglich darin, wie die Session-Id jeweils im HTTP-Request und in der HTTP-Response kodiert wird.

Cookies

- x Zur Übermittlung der Cookies werden benutzt jeder HTTP-Request und jede HTTP-Response eine gesonderte Kopfzeile.

⇒ In der HTTP-Response:

Set-Cookie: 5583

⇒ Im HTTP-Request:

Cookie: 5583

- x Der HTTP-Client (Browser) wird durch ein "Set-Cookie" aufgefordert, die nachfolgende Information lokal zu speichern und sie bei jedem weiteren HTTP-Request zum Server in einer "Cookie-Kopfzeile" mitzuschicken.*

* Anmerkung: Prinzipiell könnten hier beliebige Daten verwendet werden. In der Regel wird der Mechanismus jedoch lediglich zum Transport von Session Ids verwendet.

URL-Rewriting

- x Bei jeder HTML-Seite fügt der Server bei allen im Text enthaltenen Links eine Session-ID in die URLs ein.
- x Klickt der Anwender auf einen dieser Links, so wird die Session Id als Teil des URLs zum Server übertragen.
- x Cookies vs. URL Rewriting

Cookies vs. URL-Rewriting

- x Nicht jeder HTTP-Client unterstützt Cookies. HTTP-Clients, die Cookies unterstützen, erlauben es in der Regel dem Benutzer, den Cookie-Mechanismus auszuschalten.
- x URL-Rewriting ist für den Programmierer aufwendiger.
 - ⇒ HTML-Seiten mit Links auf andere HTML-Seiten der gleichen Web-Application müssen zum Zeitpunkt der Anfrage berechnet werden, da in alle URLs auf die eigene Web-Application Session-Ids in die URLs eingefügt werden müssen.
- x URL-Rewriting kostet serverseitig Rechenleistung.
 - ⇒ Dadurch dass an und für sich statische Seiten dynamisch, sessionspezifisch realisiert werden müssen, schränkt dies die Möglichkeiten des Caching ein. Diese Seiten können nur lokal beim Client und auch nur für die Dauer einer Session gepuffert werden. Folge: signifikant höhere Last für den Server.

Benutzerverwaltung

- x Über HTTP-Session können lediglich hintereinander ausgeführte HTTP-Messages des gleichen Clients einander zugeordnet werden.
- x Aufbauend auf dem HTTP-Session-Management kann vom Server eine Benutzerverwaltung organisiert werden.
- x Prinzip Benutzerverwaltung:
 - ⇒ Benutzer bekommen eine User-Id und einen Authentifizierungscode zugewiesen (in der Regel Passwort).
 - ⇒ Über eine "Login-Seite" der Web-Anwendung kann der Client authentifiziert werden.
 - ⇒ Durch die Authentifizierung wird serverseitig die HTTP-Session-Id der User-Id zugeordnet.
 - ⇒ Authentifizierte Benutzer erhalten Zugriff auf benutzerspezifische Daten.

5.2 Web Application Server

Statische Seiten / Dynamische Seiten

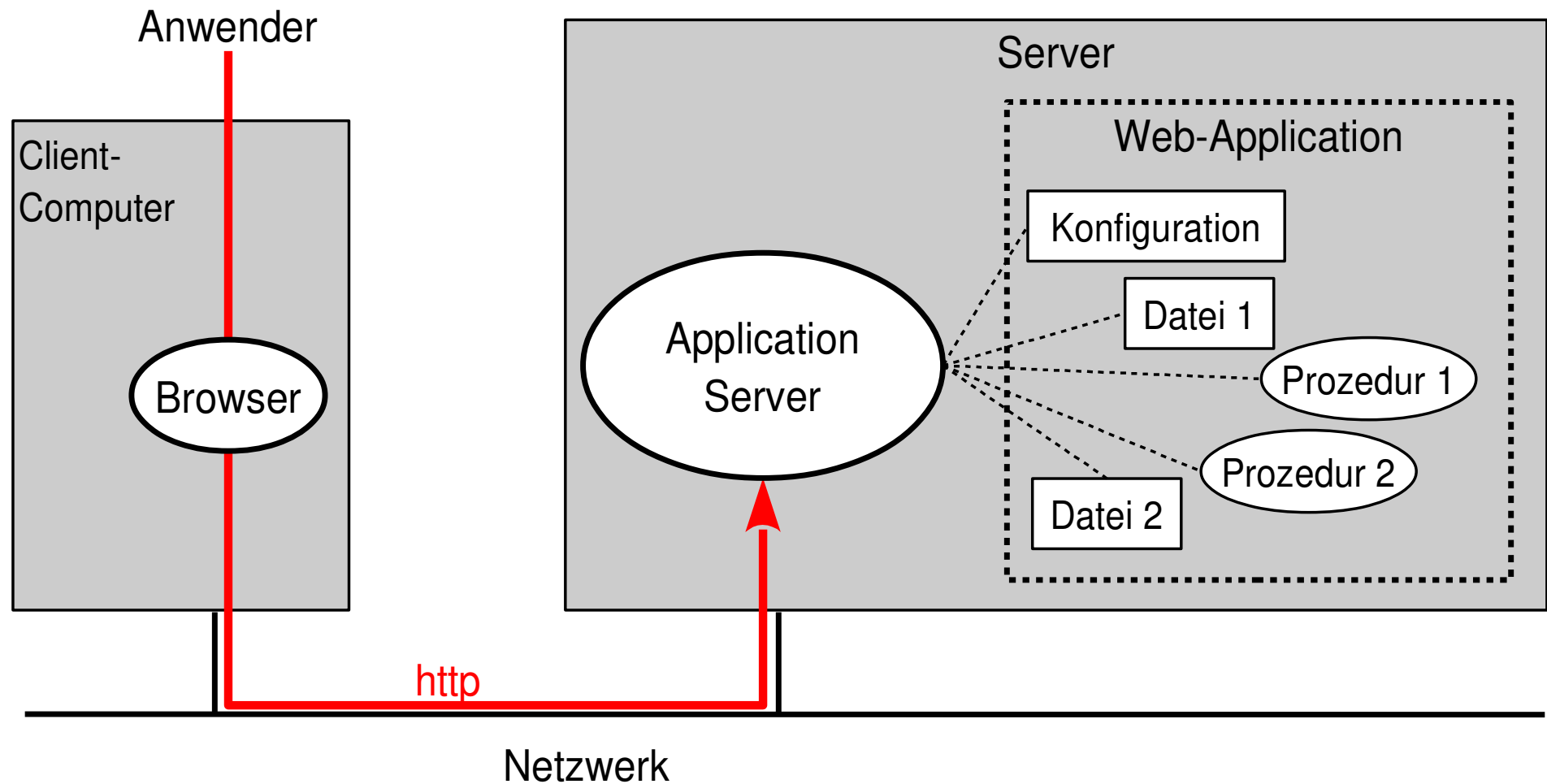
- x Web-Anwendungen setzen sich aus statischen und dynamischen Seiten zusammen. Sie bestehen aus:
 - ⇒ Dateien (statischen Seiten)
 - ⇒ Prozeduren zur dynamischen Erzeugung von Seiteninhalten
 - ⇒ eventuell Konfigurationsinformationen
- x Statische Seiten:
 - ⇒ Die Inhalte liegen bereits vor dem Eintreffen des Requests serverseitig in Form von Dateien vor. Bei einem eingehenden Request wird der Inhalt der Datei als Response zurückgeschickt.
- x Dynamischen Seiten:
 - ⇒ Die Inhalte werden serverseitig erst nach einem eingehenden Request von einem dedizierten Prozedur berechnet. In der Praxis enthalten dynamische Seiten meist HTML.

Web Application Server

Als Web Application Server bezeichnet man ein Programm, das die Aufgabe hat, Web-Anwendungen zu betreiben. Das heißt:

- x Der Web Application-Server realisiert einen HTTP-Service.
- x Durch geeignete Konfiguration wird dem Web Application-Server mitgeteilt,
 - ⇒ welche Web-Anwendung(en) er betreiben soll und wo sich diese im Dateisystem befinden
 - ⇒ welche URLs er auf welchen Dateien (statische Seiten) bzw. auf welche Prozeduren (dynamische Seiten) abbilden soll
- x Eingehende HTTP-Requests werden gemäß der Konfiguration dadurch bearbeitet, dass entweder ein bestimmter Dateiinhalt als HTTP-Response zurückgeleitet wird oder es wird eine Prozedur aufgerufen und deren Output wird als HTTP-Response an den Client zurückgeschickt.

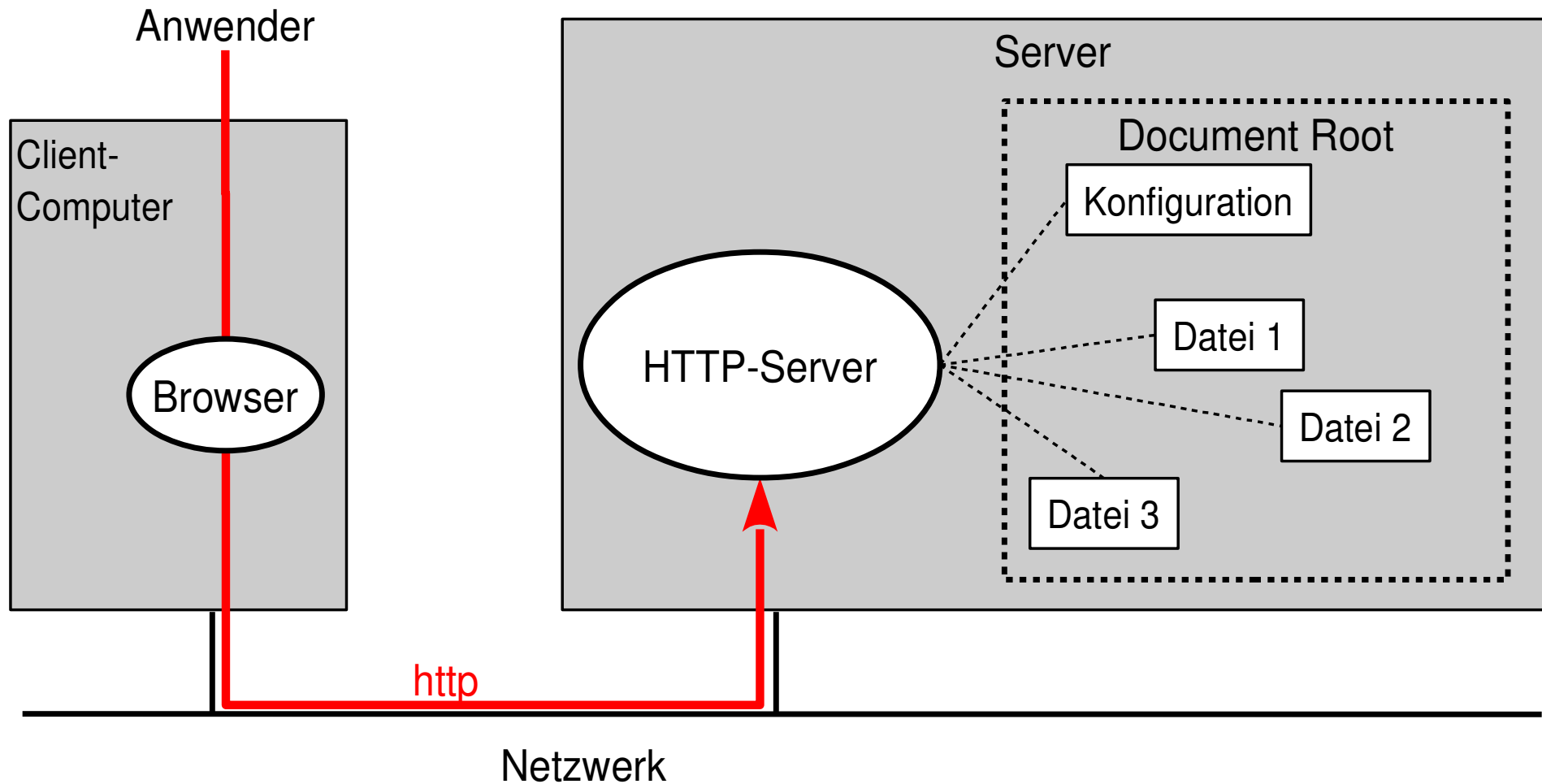
Web-Application, Application Server



HTTP-Server für statische Seiten

- x Unter einem HTTP-Server versteht man ein Programm, das einen HTTP-Service für rein statische Seiten erbringt. Anmerkung: Oft gibt es für HTTP-Server Erweiterungen für dynamische Seiten.
- x HTTP-Server für rein statische Inhalte gibt es verschiedene. Die Funktionsweise ist ähnlich.
 - ⇒ Konzept: Nach dem Start des HTTP-Serverprogramms sind die Inhalte eines Verzeichnisses, das als Document-Root bezeichnet wird, über URLs zugreifbar.
 - ⇒ Erläuterung: Bei eingehenden Request verweist der Pfad am Ende des URL auf eine Datei auf dem Server. Die Pfad-Angabe ist relativ zum Dokument-Root-Verzeichnis zu verstehen. Der Inhalt der adressierten Datei wird als Response zurückgeschickt.
 - ⇒ weltweit dominantes HTTP-Serverprogramm: Apache HTTP-Server

HTTP-Server für statische Seiten



Dynamische Web-Seiten per Socket-Programmierung?

Für dynamische Web-Seiten mag man nun auf die Idee kommen, diese direkt durch Server-Socket-Programmierung zu realisieren. Es ist dabei jedoch folgendes zu beachten:

- x Die Lösung sollte es erlauben, über mehrere TCP/IP-Verbindungen mehrere HTTP-Requests gleichzeitig zu bearbeiten (Multi-Threading).
- x Der Service sollte auch in der Lage sein, statische Seiten zur Verfügung zu stellen.
- x Man sollte nicht für jede dynamische Web-Seite einen dedizierten Port zur Verfügung stellen. Vielmehr sollte die gesamte Anwendung unter einem Service (Port) betrieben werden.
- x Die Anwendung sollte durch Änderungen in der Konfiguration auch unter https ausführbar sein.
- x Das Serverprogramm sollte das Session-Management übernehmen. ...

Web Application Server

- x Die Konzepte zur Realisierung der dynamischen Seiten sind unterschiedlich.
Beispiele:
 - ⇒ CGI-Skripte
 - ⇒ Servlets/JSPs
 - ⇒ PHP
- x Für den Anwender bedeutet dies, dass er die dynamischen Seiten auf unterschiedliche Weise programmieren muss.
- x Für die verschiedenen Konzepte dynamischer Seiten gibt es jeweils spezifische Web-Application-Server. Im Kern sind dies in der Regel HTTP-Server für rein statische Seiten mit entsprechenden spezifischen Erweiterungen für dynamische Seiten.
 - ⇒ Beispiel: PHP-Plugin für Apache

CGI-Skripte

- x Ein eingehender Request wird dadurch bearbeitet, dass das Serverprogramm ein dem URL zugeordnetes Programm startet. Dieses Programm wird als "CGI-Skript" bezeichnet.
- x Über die Standard-Eingabe des Programms bekommt es den HTTP-Request zugeführt. Die von dem Programm erzeugte Ausgabe wird als Response zum Client zurückgeleitet.

Anmerkungen zum Begriff "CGI-Skript"

- x Der Begriff CGI-Skript ist irreführend.
- x Es ist ein ganz gewöhnliches Programm gemeint, das in dem jeweiligen Betriebssystem gestartet werden kann.
- x Wie das Programm erstellt wurde und welche Programmiersprache dazu verwendet wurde, ist gleichgültig. Es muss sich nicht - wie der Name vermuten lässt - um eine Skript-Sprache handeln.
- x Der Begriff CGI (common gateway interface) bezeichnet eine Programmier-Schnittstelle mit Funktionen, die die Erstellung von CGI-Skripten hilfreich sind. Implementiert ist die CGI-Schnittstelle beispielsweise für C und PERL.

Servlets

- x Anwendungsspezifisch erstellte Java-Klassen, deren Methoden eingehende URLs bearbeiten.
- x Ein eingehender HTTP-Requests wird dadurch bearbeitet, dass er die *service*-Methode einer Instanz einer Sohnklasse von *HttpServlet* aufgerufen wird.
- x Vorteile gegenüber CGI-Skripten:
 - ⇒ Methodenaufruf - kein Programmstart
 - ⇒ Multithreading
 - ⇒ Integriertes HTTP-Session-Management

JSPs

- x HTML-Dokumente mit speziellen JSP-Tags.
- x Die JSP-Tags enthalten Java-Code. Die JSP-Tags serverseitig dynamisch durch HTML ersetzt, indem die entsprechenden Java-Methoden ausgeführt werden.
- x Vorteil: JSPs können mit einem normalen HTML-Editor visualisiert und editiert werden.
 - ⇒ Servlets sind Programme. Die Darstellung ergibt sich, wenn man sie ausführt. Das Design des Layouts zu ändern bedeutet bei Servlets, den Programmcode zu ändern.
- x JSPs werden vor dem ersten Aufruf von dem Application Server in Servlets übersetzt. Entweder beim ersten Aufruf oder schon vor der Inbetriebnahme der Web-Application.

PHP

- x Ähnlich wie JSP ist PHP eine Erweiterung von HTML um spezifische Tags, in denen dynamische Anteile der Webseite enthalten sind.
- x Die Prozeduren in den zusätzlichen Tags sind in der PHP-Skriptsprache realisiert.

5.3 Servlets, Web-Applications, Web-Application-Server

Servlet

```
@WebServlet("/HttpServlet")
```

```
public class HelloWorld extends HttpServlet {
```

```
    public void service (HttpServletRequest request, HttpServletResponse response)  
        throws IOException {
```

```
        response.setContentType("text/html");
```

```
        response.getWriter().println("Dies ist eine dynamisch erzeugte Web-Seite.");
```

```
        response.getWriter().println("<em><br/>Zeit: " +  
            new java.util.Date() + "</em>");
```

```
    }
```

```
}
```

Servlet

Servlets dienen der Bearbeitung von HTTP-Requests, deren Inhalte dynamisch - also zur Zeit des Aufrufs - erzeugt werden sollen.

- x Jedes Servlet ist einem URL zugeordnet. Die Zuordnung erfolgt mittels der Annotation `@WebServlet`.
- x Servlets sind Klassen mit einer standardisierten Schnittstelle: Sie sind Sohnklassen der Klasse *HTTPServlet*.
- x Die wichtigste Methode eines Servlets ist die Objektmethode *service*.
 - ⇒ eingehende HTTP-Requests führen zum Aufruf der *service*-Methode
 - ⇒ in der *service*-Methode wird der dynamische Inhalt der Seite erzeugt
 - ⇒ die Methode *service* wird anwendungsspezifisch überschrieben

Request - Response

- x Der Aufruf einer service-Methode dient der Bearbeitung eines HTTP-Requests: zu dem eingegangenen HTTP-Request wird eine HTTP-Response erzeugt.
- x Der service-Methode werden beim Aufruf zwei Objekte als Parameter übergeben, die sich auf den Request bzw. auf die Response des HTTP-Aufrufs beziehen.

Response

Über das Response-Objekt werden in der service-Methode Daten an den Client geschickt.

Wichtige Objektmethoden:

<code>response.setContentType(String s)</code>	Festlegen des MIME-Types der Response
<code>response.getWriter().println(String s)</code>	Nutzdaten an Client schicken

Request

Über das Request-Objekt kann die Service-Methode auf zahlreiche Request-bezogene Informationen zugreifen.

Wichtige Objektmethoden:

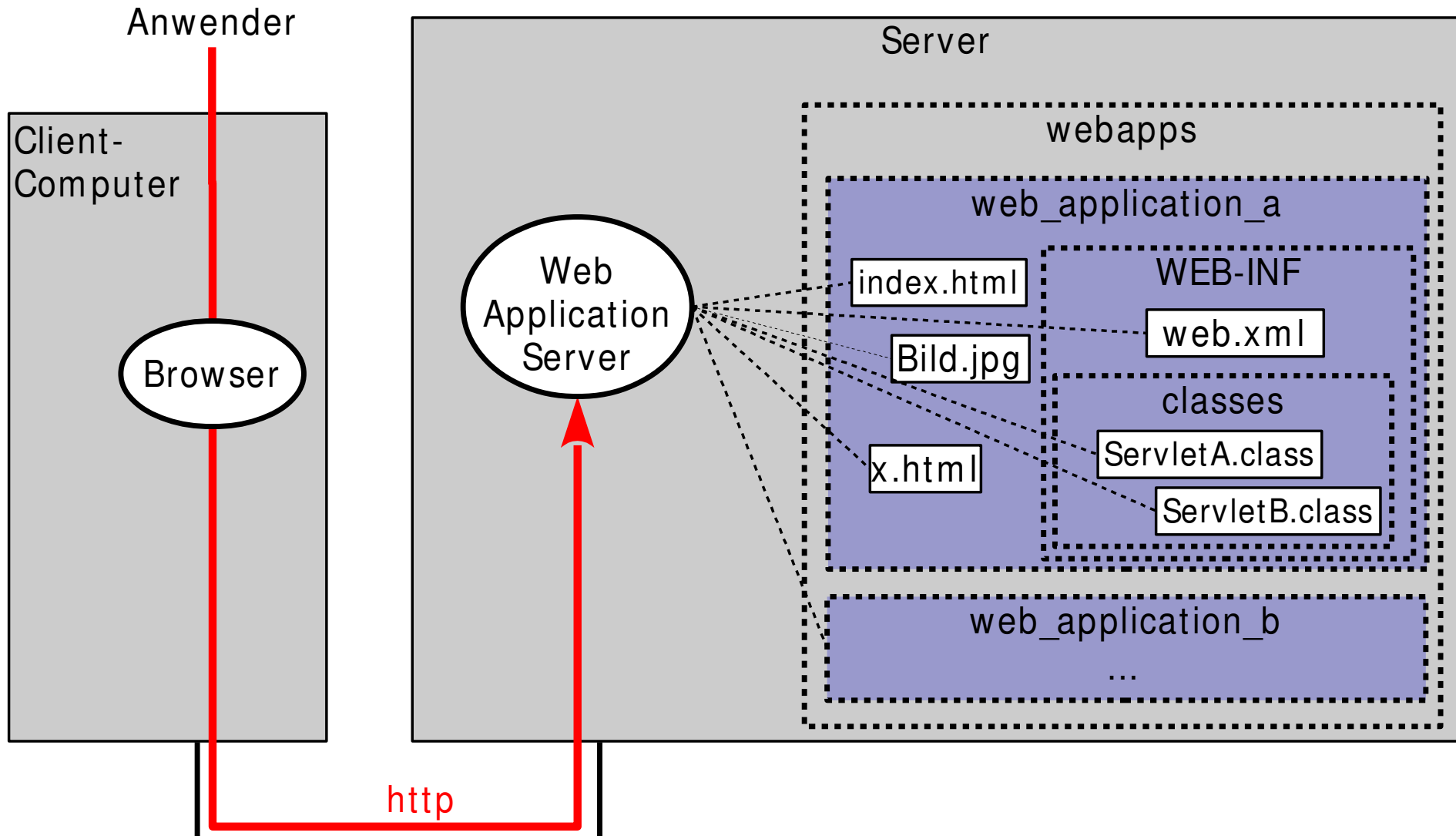
<code>request.getRequestURI()</code>	URI des Request
<code>request.getMethod()</code>	HTTP-Methode
<code>request.getSession().getId()</code>	die dem Request zugeordnete Session-Id
<code>request.getParameter(String s)</code>	Request-Parameter auslesen

Java-Web-Application

Eine Java-Web-Application ist ein Dateiverzeichnis mit einem standardisierten Aufbau. Es enthält:

- x statische Seiten
 - ⇒ verschiedene Dateien mit unterschiedlichem Typ (HTML, jpg, etc.), die über einen URL vom Client per HTTP-Get abgerufen werden können
- x die übersetzten Java-Klassen der Servlets
 - ⇒ im Unterverzeichnis WEB-INF/classes
- x einen Deployment-Descriptor mit dem Dateinamen WEB-INF/web.xml, der verschiedene Konfigurationsdaten enthält

Web-Application-Server



Web-Application-Server

Das Programm, das Java-Web-Applications betreibt, bezeichnet man als Application Server.

- x Ein Web Application Server ist ein Programm, das nach seinem Start an einem definierten Port einen HTTP-Service zur Verfügung stellt. Über diesen Service werden ein oder mehrere Web-Applications betrieben.
- x Die Verzeichnisse der Web-Applications befinden sich auf dem Server in einem dem Web Application Server bekannten Unterverzeichnis.
 - ⇒ Jede Web-Application bekommt einen Context-Path zugewiesen. In der Regel ist es das Verzeichnisname, unter dem die Web-Application auf dem Server im Web-Applications-Verzeichnis liegt. Der Context-Path ist die erste Verzeichnisebene, die im URL nach der IP-Adresse und der Port-Nummer angegeben wird.
`http://hostname:port/context-path/x/y/z.jpg`

Szenario: Bearbeitung eines HTTP-Requests

- x Der Client verbindet sich mit dem im URL angegebenen Host und Port über eine TCP/IP-Verbindung (Default-Port: 80).
- x Die Verbindung wird an den Web Application Server weitergereicht, da dieser den betreffenden Port reserviert hat.
- x Der Web Application Server entnimmt dem URL den ersten Pfad-Anteil (Context-Path) und bestimmt so, welcher Web-Application der URL zuzuordnen ist.
- x Der Web Application Server, prüft, ob dem Pfad innerhalb der Web-Application ein Servlet zugeordnet ist.
- x Verweist der URL auf ein Servlet, so wird die service-Methode des entsprechenden Servlet-Instanz aufgerufen. Gegebenenfalls muss die Servlet-Instanz zuvor erzeugt werden.
- x Ist der URL keiner Servlet-Instanz zugeordnet, so muss es sich um den Aufruf einer statischen Seite handeln. Der Inhalt der entsprechenden Datei wird dann im Rumpf der Response als Nutzdaten zum Client geschickt.

Packaging von Java Web-Applications

- x Analog zum Packaging von Libraries, werden Web-Applications mit dem jar-Mechanismus in eine gezippte Form gebracht. Für Web-Applications wird anstelle der Endung .jar die Endung .war verwendet (als Abkürzung für Web ARchive).
- x Web Application Server ermöglichen es, dass .war-Dateien im laufenden Betrieb über eine Administrationsschnittstelle hochgeladen, konfiguriert, gestartet und gestoppt werden können.

doGet, doPost, ...

- x Die in der Klasse `HTTPServlet` implementierte *service*-Methode verzweigt eingehende HTTP-Requests je nach HTTP-Methode nach `doGet`, `doPost`, ...
- x Überschreibt man, wie bisher beschrieben, die Methode `service`, so bedeutet dies, dass dieser Programmcode alle eingehenden HTTP-Requests mit beliebigen HTTP-Methoden entgegennimmt, die an das Servlet gerichtet sind.
- x Alternative:
Überschreibt man hingegen nur die Methoden `doGet` und/oder `doPost` etc., so werden durch diese Methoden nur jeweils entsprechenden HTTP-Methoden bearbeitet - HTTP-Requests mit anderen HTTP-Methoden, die an das Servlet gerichtet sind, führen zu einem Fehlercode in der Response.
- x Die Methoden `doGet`, `doPost` etc. haben die gleiche Schnittstelle wie die Methode `service`: zwei Parameter vom Typ `HttpServletRequest` und `HttpServletResponse`, kein Rückgabewert.

5.4 Request-Parameter und HTML-Formulare

Motivation

- x Mit rein statischen Seiten, die miteinander über Links verbunden sind, lassen sich einfache Web-Auftritte realisieren, die dem Benutzer Informationen zur Verfügung stellen. Die Interaktion beschränkt sich dabei darauf, dass der Benutzer über Links von Seite zu Seite springt. Der Benutzer kann darüber hinaus keine Daten an den Server übermitteln. Klassische IT-Anwendungen, die vom Benutzer eingegebene Daten in Form von Zahlen, Zeichen etc. verarbeiten, sind so nicht realisierbar.
- x Zur Verarbeitung von Benutzereingaben wird bei Web-Anwendungen gerne folgendes Schema verwendet:
 - ⇒ Der Benutzer gibt Daten in einem HTML-Formular ein.
 - ⇒ Die Daten werden beim Abschicken des Formulars als Request-Parameter an eine dynamische Seite weitergeleitet.
 - ⇒ Die dynamische Seite enthält eine Prozedur, die die Request-Parameter verarbeitet.

Request-Parameter

- x Request-Parameter sind Daten, die als Teil eines HTTP-Requests vom Client zum Server übertragen werden.
- x Jeder Request Parameter hat einen Namen und einen Wert. Sowohl der Name als auch der Wert haben den Typ String. Mit einem HTTP-Request können beliebig viele Request-Parameter mit unterschiedlichen Namen übertragen werden.
- x Es gibt zwei technische Möglichkeiten, Request-Parameter mit einem HTTP-Request zu übermitteln:
 - ⇒ Post-Parameter-Übermittlung:
Die Parameter werden durch einen Post-Request im Nutzdatenteil des Requests vom Client zum Service geschickt.
 - ⇒ Get-Parameter-Übermittlung
Die Parameter werden durch einen GET-Request als Anhang der URL übertragen. Beispiel: `http://hostname/path?x=4&y=3`

Zugriff auf Request-Parameter

Unabhängig davon, wie die Parameter übertragen wurden, kann innerhalb der service-Methode eines Servlets mit den folgenden beiden Methoden auf die Request-Parameter zugegriffen werden.

<code>request.getParameter(String s)</code>	bestimmt den Wert eines Request-Parameter, dessen Name als Parameter angegeben wird
<code>request.getParameterNames()</code>	bestimmt die Namen aller Request-Parameter

Request-Parameter und HTML-Formulare

Eine wichtige Anwendung für Request-Parameter sind Formulare:

- x Innerhalb der Formulare werden Felder definiert.
- x Jedes Feld hat einen Namen und einen Wert. Die Werte der Felder werden vom Benutzer der Seite eingegeben.
- x Durch das Betätigen eines Submit-Buttons wird ein HTTP-Request an einen definierten URL geschickt. Dabei werden die Formulardaten als Request-Parameter übertragen.

Beispiel: Formular.html, Multiplikation.java

```
Multiplikation <br\>
<form action="/MultiplikationServlet" method="get">
  a = <input type="text" name="a" width=5> <br/>
  b = <input type="text" name="b" width=5> <br/>
  <input type="submit" value="start">
</form>
```

```
@WebServlet("/MultiplikationServlet")
public class MultiplikationServlet extends HttpServlet {
    public void service (HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        int a = Integer.parseInt(request.getParameter("a"));
        int b = Integer.parseInt(request.getParameter("b"));
        response.setContentType("text/html");
        response.getWriter().println(a + "*" + b + " = " + (a*b));
    }
}
```

Beispiel-Ablauf

Multiplikation

Geben Sie bitte die beiden Faktoren.

a =

b =

$$4*8 = 32$$

Anmerkungen

- x In dem HTML-Formular wurde als "method" der Wert "get" angegeben, wodurch eine Get-Parameter-Übermittlung festgelegt wird. Alternativ könnte an dieser Stelle als Wert auch "post" angegeben werden (Post-Parameter-Übermittlung).
- x Bei dem Servlet Multiplikation wurde die service-Methode überschrieben. Das Servlet bearbeitet mit diesem Programmcode HTTP-Requests mit beliebiger HTTP-Methode und kommt somit mit beiden Arten der Parameter-Übermittlung zurecht.

5.5 Web-Application Life Cycle und Konfiguration

Web-Application Life Cycle

- x Unmittelbar nach dem Start des Application Servers startet dieser alle entsprechend konfigurierten Web-Applications. Wird der Application Server gestoppt, so stoppt dieser vorab alle Web-Applications.
- x Außerdem ermöglichen es bestimmte Application Server wie Tomcat, dass auch während des Betriebs des Application Servers einzelne Web-Applications gestartet beziehungsweise gestoppt werden.

Servlet Context Listener

x Erläuterungen:

- ⇒ Der Begriff *Servlet Context* wird in den Klassennamen von J2EE als Synonym für *Web Application* verwendet.
- ⇒ Unter einem *Listener* versteht man ein Objekt mit einer oder mehreren Methoden, das man an ein anderes Objekt übergibt mit der Absicht, dass dieses Objekt bei dem Eintreten bestimmter Ereignisse die Methoden des Listener-Objekts aufruft (zurückruft). Hier geht es um die Ereignisse "Starten der Web-Application" und "Stoppen der Web-Application".

- x Der Servlet Context Listener enthält zwei Methoden: `contextInitialized` und `contextDestroyed`, die unmittelbar vor dem Starten der Web-Application bzw. unmittelbar nach dem Stoppen der Web-Application ausgeführt werden.

... Servlet Context Listener

Will man unmittelbar vor dem Start und/oder unmittelbar nach dem Stoppen einer Web-Application Programmcode ausführen, so muss man eine Sohnklasse von `ServletContextListener` (ein Interface) deklarieren, dort die beiden Methoden `contextInitialized` und `contextDestroyed` überschreiben und der Sohnklasse die Annotation `@WebListener` voranstellen.

Schema:

```
@WebListener
public class MyServletContextListener implements ServletContextListener{
    public void contextInitialized(ServletContextEvent sce) {
        // hier steht Programmcode, der vor dem Starten ausgeführt werden soll
    }
    public void contextDestroyed(ServletContextEvent arg0) {
        // hier steht Programmcode, der nach dem Stoppen ausgeführt werden soll
    }
}
```

Anwendungsszenarien für Servlet Context Listener

- x Soll innerhalb der Web-Application auf einen anderen Service zugegriffen werden, so kann innerhalb der *contextInitialized*-Methode die Verbindung zu dem Service aufgebaut und in der *contextDestroyed*-Methode die Verbindung wieder abgebaut werden.
- x Analoges Vorgehen, wenn während des Betriebs einer Web-Application andere Ressourcen benötigt werden: Reservieren der Ressource in der Methode *contextInitialized*, Freigabe in der Methode *contextDestroyed*.
- x Werden einzelne Objekte über die gesamte Laufzeit einer Web-Application hin benötigt, so können diese vorab in der *contextInitialized*-Methode erzeugt werden.
- x Sollen vor Beginn bestimmte "Initialisierungs-Methoden" des Programmcodes aufgerufen werden (z.B. zum Auslesen von Daten aus Dateien), so kann man diese Aufrufe in der *contextInitialized*-Methode unterbringen.

Context Parameter

- x Unter Context Parametern versteht man Parameter, die innerhalb des Deployment Descriptors definiert werden und auf die innerhalb der Web-Application lesend zugegriffen werden kann.
- x Jeder Context Parameter hat einen Namen - eine Zeichenkette vom Typ String.
- x Jeder Context Parameter hat einen Wert vom Typ String.
- x Jeder Context Parameter wird im Deployment Descriptor als ein Paar bestehend aus Namen und Wert definiert.
- x Innerhalb der Web-Application kann über den Namen auf den Wert des Context Parameters zugegriffen werden.

Beispiel: Deklaration von Context Parametern

```
<context-param>  
  <param-name>Portalbetreiber</param-name>  
  <param-value>Franz Müller und Söhne GmbH</param-value>  
</context-param>  
  
<context-param>  
  <param-name>Portalbetreiber-Adresse</param-name>  
  <param-value>Wilhemstr. 182, 66213 Münchfurt</param-value>  
</context-param>
```

Zugriff auf Context Parameter

- x Innerhalb der Web-Application wird über ein ServletContext-Objekt auf die Context Parameter zugegriffen.
- x Auf das ServletContext-Objekt einer Web-Application kann aus verschiedenen Stellen in der Web-Application zugegriffen werden.

Beispiele:

⇒ In der service-Methode eines Servlets:

```
ServletContext scontext = getServletContext(); ...
```

⇒ In der contextInitialized-Methode eines Context-Listeners:

```
public void contextInitialized(ServletContextEvent sce) {  
    ServletContext scontext = sce.getServletContext();  
    ...  
}
```

... Zugriff auf Context Parameter

- x Objekte vom Typ ServletContext stellen unter anderem zwei Methoden für den Zugriff auf Context-Parameter bereit: *getInitParameter* und *getInitParameterNames*.

scontext.getInitParameter(String s)	bestimmt den Wert eines Context-Parameters, dessen Name als Parameter angegeben wird
scontext.getInitParameterNames()	bestimmt die Namen aller Context-Parameter

Wozu Context Parameter?

- x Damit eine einmal erstellte Web-Application flexibel an unterschiedliche Anforderungen angepasst werden kann, werden die entsprechenden Konfigurations-Einstellungen als Parameter realisiert.
- x Eine einmal erstellte, fertig übersetzte Web-Application kann so noch vor der Inbetriebnahme konfiguriert werden, indem man mit einem Editor im Deployment-Descriptor die Werte Context Parameter anpasst. Es reicht, wenn dazu an den Kunden Byte-Code ausgeliefert wird - kein Source Code (Programmtext).
- x Hätte man die entsprechenden Werte fest in den Programmcode geschrieben, so müsste man den Source-Code mit an den Kunden ausliefern. Der Kunde müsste vor der Inbetriebnahme der Web-Application den Source Code an der entsprechenden Stelle ändern und anschließend den Java-Programmcode neu übersetzen.

5.6 Server Side Includes, Forwarding, Redirecting

Server Side Includes

- x Bei der Bearbeitung eines HTTP-Requests können in die Response auch Inhalte anderer URLs des gleichen Servers eingebunden werden.

```
request.getRequestDispatcher(String path).include(request,response);
```

- x Auf diese Weise können in eine HTML-Seite sowohl statische als auch dynamische Inhalte eingebunden werden.
- x Das Konzept eignet sich nur für Inhalte der gleichen Web-Application.
- x Sollen Web-Inhalte beliebiger Web-Seiten eingebunden werden, so ist es prinzipiell auch möglich, dass das Servlet in der Rolle eines HTTP-Clients Daten von einem beliebigen Server abruft.
 - ⇒ Server Side Includes sind jedoch bei weitem effizienter. Das Einbinden der Inhalte erfolgt durch einen einfachen Methodenaufruf - es muss keine TCP/IP-Verbindung aufgebaut werden.

Forwarding

- x Mit dem nachfolgenden Befehl wird ein anderer URL der gleichen Web-Anwendung mit der Bearbeitung des Requests beauftragt.

```
request.getRequestDispatcher(String path).forward(request,response);
```

- x Anders als bei include(), ergibt sich der Inhalt der Response alleine durch die Bearbeitung des forward-Aufrufs. Alle anderen zuvor an die Response geleiteten Ausgaben werden ignoriert. Die Ausgabe wird nach Aufruf des Befehls geschlossen und es kann nachfolgend nichts angehängt werden.
- x Aus Client-Sicht ändert sich die URL nicht. Das Ergebnis erscheint im Browser mit der URL, unter der der forward-Befehl aufgerufen wurde.

Redirecting

- x Der Aufruf der Methode

```
response.sendRedirect("https://www.hs-furtwangen.de/studium");
```

- x führt zu folgender HTTP-Response

```
HTTP/1.1 302 Moved Temporarily  
Location: http://127.0.0.1:8080/OPWebApp/x.html  
Content-Type: text/plain  
Content-Length: 0  
Date: Sun, 14 Dec 2014 21:48:04 GMT  
Server: Apache Coyote/1.0
```

- x Durch diese spezielle HTTP-Response wird der Browser aufgefordert, einen zweiten HTTP-Request zu schicken, und dessen HTTP-Response anzuzeigen.

Redirecting und Forwarding

- x Die Auswirkung von Redirecting gleicht der des Forwarding: Ein anderer URL des gleichen Servers berechnet die Response.
- x Anders als beim Forwarding, ändert sich der Pfad beim Redirecting im Browser und zeigt anschließend auf den Pfad, der den Request bearbeitet hat: den Parameter von `sendRedirect()`.
- x Zu Beachten ist ferner, dass bei `sendRedirect` absolute Pfadangaben immer als ersten Pfadanteil den Context-Path der Web-Application enthalten müssen.
- x Bei Server Side Includes und beim Forwarding bezieht sich eine absolute Pfadangabe immer auf die aktuelle Web-Application. Der Context-Path ist nicht Bestandteil der Pfadangabe.

5.7 HTTP-Sessions

Session-IDs

- x Der Application Server ordnet allen bei ihm eingehenden HTTP-Requests Sessions-IDs zu.
- x Bei den Session-IDs handelt es sich um kurze Strings, die in der Regel eine Zahl repräsentieren.
 - ⇒ Tomcat: 32 Hexadezimalzeichen (16 Byte)
- x Die Session-ID wird in aufeinander folgenden Requests zwischen Client und Application Server hin- und hergeschickt. Wie der Application Server Sessions verfolgt, Cookies oder URL-Rewriting, kann für jede Web-Application konfiguriert werden.
- x Während der Bearbeitung eines HTTP-Requests kann das Servlet auf die Session-ID zugreifen.

```
String sid = request.getSession().getId();
```

URL-Rewriting

- x Standardmäßig basiert das Session-Management auf Cookies.
- x Cookie-basiertes Session-Management kann unterbunden werden
 - ⇒ durch eine entsprechende serverseitige Konfiguration
 - ⇒ wenn der Browser Cookies ablehnt (Browser-Einstellung)
- x Anstelle von Cookie-basiertem Session-Management kann auch auf URL-Rewriting ausgewichen werden.
- x In diesen Fällen müssen bei allen vom Server ausgelieferten HTML-Seiten jeweils in alle URL-Links Session-Ids hineincodiert werden. Konsequenz:
 - ⇒ Die HTML-Seiten, die Verweise auf andere HTML-Seiten des Servers haben, müssen dynamisch erzeugt werden.
 - ⇒ Weitgehender Verzicht auf statische HTML-Seiten!

... URL-Rewriting

- x Zur Codierung von URLs steht die folgende Methode der Klasse HttpServletResponse zur Verfügung:

```
response.encodeURL(String s)
```

- x Beispiel:

⇒ URL vor der Codierung

```
http://127.0.0.1:8080/ContextPath/path?x=3&y=5
```

⇒ URL nach der Codierung

```
http://127.0.0.1:8080/ContextPath/path;jsessionid=CF0E1B9E825E4F?x=3&y=5
```

... URL-Rewriting

- x Erlaubt der Browser Cookies und sind Cookies auch serverseitig zugelassen (Konfiguration des Application Servers), so gibt die Methode `encodeURL` den String `s` unverändert zurück.
- x Es gibt konzeptionell zwei Möglichkeiten Web-Applications zu realisieren: mit und ohne URL-Rewriting.
- x Die Entscheidung für oder gegen URL-Rewriting zieht sich durch die große Teile der Web-Anwendung und sollte deshalb vor Beginn der Implementierungsarbeiten einer Web-Application getroffen werden.

Web-Applications mit und ohne URL-Rewriting

<i>ohne URL-Rewriting</i>	<i>mit URL-Rewriting</i>
In den von der Web-Application erzeugten HTML-Seiten werden die URL-Links, die auf interne Seiten verweisen, <u>nicht</u> mit encodeURL codiert.	In den von der Web-Application erzeugten HTML-Seiten werden die URL-Links, die auf interne Seiten verweisen, mit encodeURL codiert.
Statische HTML-Seiten sind auch dann möglich, wenn Sie interne Links enthalten.	HTML-Seiten mit internen Links müssen dynamisch erzeugt werden.
Unterstützt der Browser keine Cookies, so ist kein Session-Management möglich.	Unterstützt der Browser keine Cookies, so wird automatisch auf URL-Rewriting zur Session-Verfolgung umgeschaltet.

5.8 Attribute und Scopes

Wo die Daten abspeichern?

- x Innerhalb einer Web-Application können die Daten an verschiedenen Stellen abgespeichert werden.
- x Beispiele:
 - ⇒ lokale Variablen der service-Methode
 - ⇒ Objektattribute der Servlet-Instanzen
 - ⇒ Klassenattribute

Attribute

Für Web-Applications steht ein einheitlicher Mechanismus für gemeinsam genutzte Daten zur Verfügung: Attribute.

- x Attribute sind Paare bestehend aus einem Attributnamen und einem Attributwert.
- x Der Attributname ist ein String.
- x Der Attributwert hat den Typ Object.
- x Auf Attribute kann lesend und schreibend zugegriffen werden.
- x Java unterscheidet verschiedene Arten von Attributen, je nachdem, für welchen Bereich (Scope) sie innerhalb der Web-Application eingesetzt werden sollen. Für jeden Scope stehen eigene Attribute zur Verfügung, auf die über Objektmethoden unterschiedlicher Objekte zugegriffen wird. Die Objektmethoden heißen jedoch jeweils gleich und folgen den gleichen Prinzip.

... Attribute

- x Es gibt in vier Scopes: Application Scope, Session Scope, Request Scope und Page Scope
- x Entsprechend bezeichnet man die jeweiligen Attribute als Application-Attribute, Session-Attribute, Request-Attribute und Page-Attribute.
- x Diese Attribute sind nicht zu verwechseln mit Objektattribute und Klassenattribute. Vergleich:

Objektattribut Klassenattribut	Objekt- bzw. Klassenbezogene Variablen in Java.
Application-Attribut Session-Attribut Request-Attribut Page-Attribut	Speicherplätze für gemeinsam genutzte Daten in Web-Applications, auf die mit besonderen Methoden zugegriffen werden kann.

Scopes

Application Scope	<p>Attribute im Application-Scope sind der gesamten Web-Application zugeordnet. Sie dienen der Aufbewahrung von "Globalen Daten", auf die überall in der Web-Application zugegriffen werden kann.</p> <p>Tipp: Initialisierung dieser Attribute mit Hilfe eines Servlet Context Listeners.</p>
Session Scope	<p>Daten, die sich auf eine HTTP-Session beziehen, können in Session-Attributen abgespeichert werden.</p> <p>Prinzip: Während eines HTTP-Request werden in einem Session-Attribut Daten abgelegt. Bei einem nachfolgenden HTTP-Request der gleichen Session kann unter dem zuvor verwendeten Attributnamen wieder auf die Daten zugegriffen werden. Die unter einem Attributnamen abgelegten Daten sind spezifisch für die Session - unter dem gleichen Attributnamen werden aus einer anderen Session heraus andere Daten abgelegt.</p>

... Scopes

Request Scope	<p>Die Daten in den Request-Attributen sind spezifisch einem HTTP-Request zugeordnet und existieren auch nur für die Dauer eines HTTP-Requests.</p> <p>Anwendung: Ein Servlet ruft ein anderes Servlet mit forward oder include auf und übergibt dem aufgerufenen Servlet Daten über die Request-Attribute.</p>
Page Scope	<p>Daten, die nur innerhalb einer Seite und nur für die Dauer der Bearbeitung der Seite verwendet werden sollen. Diese Daten sind aus Seiten, die mit include oder forward aufgerufen wurden, nicht sichtbar.</p> <p>Diese Attribute sind nur in Zusammenhang mit JSPs von Bedeutung, worauf später eingegangen werden soll.</p>

Zugriffsmethoden für Attribute

Für jeden Scope stehen spezifische Methoden zur Verfügung, mit denen auf die Attributwerte zugegriffen werden kann. Die Namen und die Funktion dieser Methoden sind für alle Scopes einheitlich:

Object <code>getAttribute(String s)</code>	Bestimmt den Wert des Attributes mit dem Namen <code>s</code> . Existiert das Attribut nicht, so ist der Rückgabewert null.
void <code>setAttribute(String s, Object o)</code>	Einem Attribut mit dem Namen <code>s</code> wird der Wert <code>o</code> zugeordnet. Existiert das Attribut noch nicht, so wird es hiermit erzeugt.
Enumeration <code>getAttributeNames()</code>	bestimmt die Namen aller Attribute des Scopes

Zugriff auf Attributwerte

Je nach Scope befinden sich die Zugriffsmethoden in unterschiedlichen Objekten.

Scope	Klasse des Objekts mit den Zugriffsmethoden	Zugriff auf das Objekt
Application Scope	ServletContext	Objektmethode getServletContext() der Klasse HttpServlet. Objektmethode getServletContext() der Klasse ServletContextEvent (Parameter der Servlet Context Listener Methoden).
Session Scope	HTTPSession	Objektmethode getSession() der Klasse HttpServletRequest - ein Parameter der service-Methode des Servlets.
Request Scope	HttpServletRequest	Parameter der service-Methode des Servlets.
Page Scope	PageContext	-

Erläuterungen

- x Session-Attribute basieren auf dem Session-Management:
 - ⇒ Der Application Server ordnet jeder HTTP-Session eine eindeutige Session-Id zugeordnet.
 - ⇒ Session-Attribute sind einer HTTP-Session mit einer eindeutigen Session-Id zugeordnet.
- x Ein Session-Attribut mit dem Namen "x" existiert zu einem Zeitpunkt in der Regel mehrfach: Es kann einmal in jeder Session existieren und und kann dort jeweils einen anderen Wert haben. Wird bei der Bearbeitung eines Requests innerhalb der Service-Methode auf das Attribut "x" zugegriffen, so erhält man das Session-Attribut der "eigenen" Session, also der Session, die dem aktuell bearbeiteten Request zugeordnet ist.
- x Analog dazu können gleichzeitig gleichnamige Request-Attribute und Page-Attribute existieren.

Beispiel für den Einsatz von Session-Attributen

Aufgabenstellung:

- x Über Servlet-Aufrufe aus HTML-Formularen sollen Waren zu einem Warenkorb hinzugefügt werden.

Lösungsansatz:

- x Benötigt wird zunächst eine "Warenkorb-Klasse", in deren Instanzen die von den Benutzern ausgewählten Produkte gespeichert werden. Die Klasse Warenkorb enthält u. a. eine Methode zum Hinzufügen von Artikeln.
- x Wenn der Anwender einen Artikel anklickt, wird ein Servlet aufgerufen, das die Operation wie folgt bearbeitet:
 - ⇒ Prüfen, ob bereits ein Warenkorb-Objekt (cart) als Session-Attribut existiert. Gegebenenfalls einen neuen, leeren Warenkorb als Instanz der Klasse Warenkorb erzeugen und als Session-Attribut in der Session speichern. Die Hinzufügen-Methode des Warenkorbs aufrufen.

... Beispiel für den Einsatz von Session-Attributen

```
public void service (HttpServletRequest request, HttpServletResponse response)
    throws IOException {
    response.setContentType("text/html");
    Vector cart = (Vector)request.getSession().getAttribute("cart");
    if (cart == null) {
        cart = new Vector();
        request.getSession().setAttribute("cart",cart);
    }
    String item = request.getParameter("item");
    if (item != null)
        cart.add(item);
    response.getWriter().println("Inhalt des Einkaufskorbs:<br\\>");
    for (int i=0; i<cart.size(); i++) {
        response.getWriter().println(" - " + cart.elementAt(i) + "<br\\>");
    }
}
```

Attribute vs. Parameter

- x Neben den in diesem Abschnitt behandelten Attributen wurden zwei Arten von Parametern behandelt: Request Parameter und Context Parameter.
- x Die Begriffe Attribute und Parameter stehen bei Web-Applications für unterschiedliche Konzepte.
- x Attribute und Parameter stehen jeweils für Werte, die mit einem Namen in Form eines Strings bezeichnet werden.
- x Parameterwerte kommen von externen Quellen (Formular, Deployment Descriptor) und haben den Typ String. Innerhalb der Web-Application kann nur lesend auf die Parameter zugegriffen werden.
- x Die Attribute haben Werte vom Typ Object. Auf Attributwerte kann innerhalb der Web-Application lesend und schreibend zugegriffen werden.

5.9 Zugriffskontrolle, Benutzerverwaltung

Benutzer und Rollen

- x Der Application Server verwaltet eine Benutzerliste, wobei für jeden Benutzer die folgenden Daten gespeichert sind:
 - ⇒ der Benutzername
 - ⇒ das Passwort
 - ⇒ ein oder mehrere ihm zugeordnete Rollen
- x Diese Liste bezieht sich auf alle Web-Applications, die auf dem Application Server ausgeführt werden sollen.
- x In den Deployment-Descriptors der Web-Applications können rollen-spezifische Zugriffsbeschränkungen für bestimmte URLs definiert werden.
 - ⇒ Schema: nur Rolle X darf auf Pfad /y zugreifen

... Benutzer und Rollen

- x Benutzer, die auf die zugriffsbeschränkten Seiten zugreifen wollen, werden durch den Application Server aufgefordert, sich zu authentifizieren. Erst nach einer erfolgreicher Authentifizierung und nachdem geprüft wurde, ob der Benutzer die benötigte Rolle zugewiesen wurde, wird der Benutzer auf die entsprechenden Seiten weitergeleitet.

Benutzer und Rollen in Tomcat

Die Datei tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="kunde"/>
  <role rolename="verkäufer"/>
  <role rolename="administrator"/>
  <user username="maier" password="x2f" roles="kunde"/>
  <user username="müller" password="zeitlos" roles="verkäufer,kunde"/>
  <user username="tim" password="mit" roles="kunde"/>
  <user username="fritz" password="zeitlos" roles="administrator"/>
</tomcat-users>
```

Zugriffsbeschränkungen

Deployment Descriptor (WEB-INF\web.xml)

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>securearea</web-resource-name>
    <url-pattern>/secure/*</url-pattern>
  </web-resource-collection>
  <auth-constraint><role-name>kunde</role-name></auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config><auth-method>BASIC</auth-method></login-config>
<security-role><role-name>kunde</role-name></security-role>
```

Basic Authentication

Der Anmelde-Vorgang des Benutzers hat zur Folge, dass in den nachfolgenden HTTP-Requests eine authorization-Zeile im Request-Header mitgeschickt wird. Sie enthält base64-codiert (unverschlüsselt!) die User-Id und das Passwort:

```
authorization : Basic cm9vdDpwYXNzd29yZA==
```

Authentifizierte Benutzer

- x Innerhalb der Servlets kann bei der Bearbeitung eines HTTP-Requests auf den Benutzernamen eines authentifizierten Benutzers zugegriffen werden:

```
request.getRemoteUser()
```

⇒ Im Fall eines nicht authentifizierten Benutzers ergibt sich als Rückgabewert null.

- x Darauf aufbauend können für authentifizierte Benutzer benutzerspezifische Operationen ausgeführt werden.

⇒ Beispiel:

Dem authentifizierten Benutzer werden die Umsätze seines Kontos angezeigt. Der Benutzername `request.getRemoteUser()` dient als Schlüssel beim Zugriff auf die entsprechende Datenbanktabelle.

... Authentifizierte Benutzer

- x Der Einsatz der Benutzer-Authentifizierung hat somit zwei Implikationen:
 - ⇒ Der Zugriff auf bestimmte URLs kann durch eine entsprechende Einstellung im Deployment-Descriptor auf ausgewählte Benutzer beschränkt werden.
 - ⇒ Bei der Bearbeitung von HTTP-Requests können benutzerspezifische Informationen bearbeitet werden.

Form-Based Authentication

Neben der beschriebenen "Basic-Authentication" können Web-Applications im Deployment-Descriptor auch so konfiguriert werden, dass sie eine Form-Based-Authentication durchführen.

- x User-Id und Passwort werden dabei über ein HTML-Formular eingegeben.
- x Die HTML-Seite mit dem Formular kann frei gestaltet werden und auch die Fehlerseite, die nach einer fehlgeschlagenen Anmeldung angezeigt wird.

Sicherheit

- x Moderne Webportale arbeiten in der Regel durchgängig mit HTTP und nicht mehr mit HTTPS. Ein wesentlicher Grund dafür ist es, dass mit HTTPS die Authentizität der Webseite sichergestellt werden kann.
- x Wenn man mit geschützten Bereichen arbeitet, so empfiehlt es sich unbedingt mit HTTPS und nicht mit HTTP zu arbeiten.
 - ⇒ Durch die Zugangsbeschränkung möchte man den Zugriff auf einen bestimmten geschützten Bereich auf eine bestimmte Benutzergruppe beschränken. HTTP bietet jedoch keine Sicherheit dafür, dass die Kommunikation abgehört wird und so ein Angreifer auf dem Transportweg die geschützten Daten mitliest und darüber hinaus auch die bei der Authentifizierung verschickten Benutzernamen und Passwörter.

Zugriffsbeschränkungen mit HTTPS

Deployment Descriptor (WEB-INF\web.xml)

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>securearea</web-resource-name>
    <url-pattern>/secure/*</url-pattern>
  </web-resource-collection>
  <auth-constraint><role-name>kunde</role-name></auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

⇒ Zugriffe auf den Pfad /secure/* werden auf den HTTPS-Port umgeleitet.

5.10 Welcome Files, Error Pages, Mime Types

Welcome Files

Im Deployment Descriptor kann festgelegt werden, wie ein URL bearbeitet werden soll, der nur den Context-Pfad der Web-Applikation und keine weiteren Pfad- oder Dateiangaben enthält. Beispiel:

```
http://127.0.0.1:8080/OPWebApp
```

Die nachfolgende Deklaration im Deployment Descriptor legt fest, dass dieser Request auf index.html umgeleitet wird. Sollte index.html nicht existieren, so wird stattdessen index.htm angesprochen, und sollte auch diese Datei nicht existieren, dann index.jsp.

```
<welcome-file-list>  
  <welcome-file>index.html</welcome-file>  
  <welcome-file>index.htm</welcome-file>  
  <welcome-file>index.jsp</welcome-file>  
</welcome-file-list>
```

Error Pages

- x Zur Bearbeitung von HTTP-Requests ruft der Application Server die service-Methode eines Servlets auf.
- x Kommt es innerhalb der service-Methode zu einer Exception und wird diese Exception nicht innerhalb der service-Methode abgefangen wird, so wird diese durch den Application Server abgefangen.
 - ⇒ Es ist also nicht etwa so, dass eine Exception innerhalb eines Servlets zum Absturz des Hauptprogramms, also des Application Servers, führen würde.
- x Nachdem der Application Server die Exception abgefangen hat, beantwortet er den HTTP-Request mit einer Error Page - einer HTML Seite, die den Anwender darüber informiert, dass die Bearbeitung des Requests gescheitert ist. Der Head der Response enthält dann den Fehlercode 500.

HTTP/1.1 500 Internal Server Error

Die Standard-Error-Page

Soweit nicht anders konfiguriert, erscheint nach einer Exception folgende Error Page:

Konfiguration von eigenen Error Pages

- x Im Deployment Descriptor können für eine Web Application auch andere Error Pages konfiguriert werden.
- x Ein Error-Page-Eintrag im Deployment Descriptor bestimmt, welche Error Page bei einer bestimmten Exception aufgerufen werden soll.

```
<error-page>  
  <exception-type>java.lang.NullPointerException</exception-type>  
  <location>/errorpage.html</location>  
</error-page>
```

- x Dieser Eintrag bezieht sich auf alle Servlets der Web-Application.
 - ⇒ Im Beispiel: Wenn es in irgend einem Servlet der Web-Application zu einer NullPointerException kommt, so wird dem Client der Inhalt der Datei errorpage.html angezeigt.

Anmerkung zu Error Pages

- x Anstelle den soeben beschriebenen Error-Page-Mechanismus zu verwenden, könnte man alternativ auch innerhalb der service-Methode der Servlets die Exceptions abfangen, um in der Exception-Behandlung ein `sendRedirect` auf die jeweilige Fehlerseite auszuführen.
- x Der Effekt wäre aus Sicht des Anwenders der gleiche.
- x Enthält Ihre Web-Application jedoch mehrere Servlets und sollen bestimmte Exceptions in all diesen Servlets in einheitlicher Weise bearbeitet werden, so empfiehlt sich der Error-Page-Mechanismus.

MIME Types und Servlets

- x Um den Daten im Rumpf einer HTTP-Response ein Datenformat zuzuordnen, wird im Header der Response eine Content-Type-Zeile übertragen, die den MIME-Type enthält.

⇒ Beispiel:

```
Content-Type: text/html
```

- x Mit der Methode `setContentType` wird innerhalb der `service`-Methode eines Servlets der Typ MIME Type der Response festgelegt.

```
response.setContentType("text/html");
```

MIME Types und statische Seiten

- x Bei den statischen Inhalten weist der Application Server den Daten einen MIME-Type aufgrund der Dateiendung zu. Für diese Zuordnung haben sich Standards etabliert, und an diesen orientiert sich der Application Server.
- x Soll von diesen Standards abgewichen werden oder soll ein neuartiges Datenformat mit einer spezifischen Endung zum Einsatz kommen, so lässt sich dies im Deployment-Descriptor durch ein spezifisches MIME-Mapping, einer Zuordnung zwischen Datei-Endung und MIME-Type, definieren.

Beispiel

- x Das folgende MIME-Mapping im Deployment Descriptor legt fest, dass Dateien mit der Endung .html als unformatierter, reiner Text betrachtet werden sollen.

```
<mime-mapping>  
  <extension>html</extension>  
  <mime-type>text/plain</mime-type>  
</mime-mapping>
```

- x Konsequenzen:
 - ⇒ Bei einem HTTP-Request, dessen URL eine statische Seite mit der Endung .html adressiert, wird der Application Server eine HTTP-Response erzeugen, bei der der Content-Type-Eintrag den MIME-Type text/plain angibt.
 - ⇒ Der Browser wird die ankommenden Daten nicht als HTML interpretieren und sie unformatiert als reinen Text darstellen.

5.11 JSPs - Motivation

Beispiel - Formularseite

Formularseite


← → ↻ 🏠 ⓘ |Formularseite. ⌵ ⋮ 📌 ⭐ ⬇️ ⏏️

Kann denn Sparen Sünde sein?

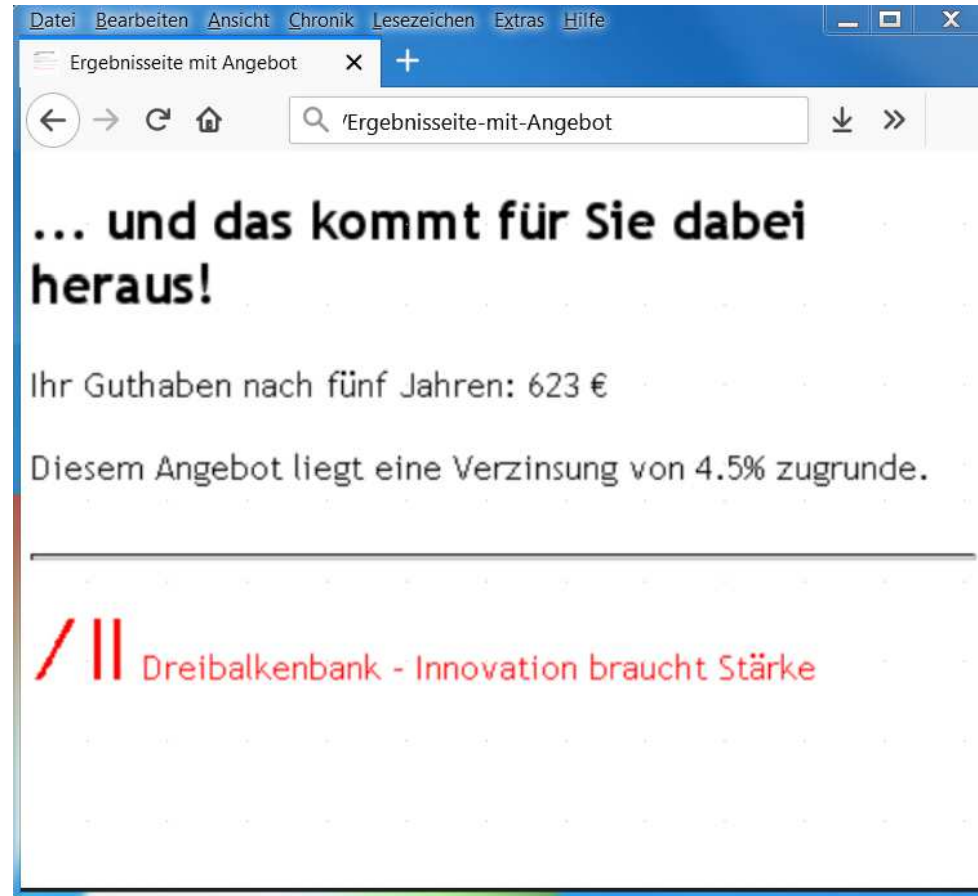
Möglichkeiten zur Geldanlage gibt es viele. Wenn Sicherheit für Sie ein Thema ist, dann sind Sie bei der Dreibalkenbank mit Ihrem innovativen Festgeldangebot genau richtig.

Sie wollen wissen, wie weit Sie mit einer Festgeldanlage in fünf Jahren kommen? Testen Sie uns!

Ihr Anlagebetrag: €

 Dreibalkenbank - Innovation braucht Stärke

Beispiel – Ergebnisseite



Statischer Anteil - dynamischer Anteil

The screenshot shows a web browser window with the title 'Ergebnisseite mit Angebot'. The address bar shows the URL 'Ergebnisseite-mit-Angebot'. The page content is highlighted in blue. The text on the page includes: '... und das kommt für Sie dabei heraus!', 'Ihr Guthaben nach fünf Jahren: 623 €', 'Diesem Angebot liegt eine Verzinsung von 4.5% zugrunde.', and the Dreibalkenbank logo with the tagline 'Innovation braucht Stärke'. Two arrows point to specific parts of the page: one from the text 'statischer Anteil' to the first blue highlighted section, and another from the text 'dynamischer Anteil' to the orange box containing the value '623 €'.

statischer Anteil

dynamischer Anteil

Motivation

- x Will man in einer Web-Anwendung nicht nur statische, vorab definierte Inhalte präsentieren, sondern eine weitergehende Interaktion mit dem Benutzer ermöglichen, so benötigt man dynamische Web-Seiten.
- x Die Seiten einer Web-Application müssen in gestalterischer Hinsicht bearbeitet werden. Alle Seiten, statische Seiten und dynamische Seiten, sollten ein einheitliches Layout erhalten.
- x Statische Seiten lassen sich in einfacher Weise editieren:
 - ⇒ einfache Text-Editoren
 - ⇒ spezialisierte HTML-Editoren (WYSIWYG)

... Motivation

- x Bei Servlets ergibt sich der Inhalt der Seite durch das Ausführen der service-Methode.
 - ⇒ Den Aufbau der vom Servlet erzeugten HTML-Seite zu ändern, bedeutet, das Programm zu ändern!
- x Es ist nicht möglich, Servlets in einem WYSIWYG-Editor zu editieren.
- x In der Regel sind bei einer Web-Seite nur einzelne Teilstücke dynamisch zu berechnen.

JSP (Java Server Page)

Idee: Seiten mit dynamischen Teilstücken

- x JSP-Datei: Eine Seite in Form einer Textdatei (in der Regel HTML), die an verschiedenen Stellen so genannte "JSP-Tags" enthält.
- x Ein JSP-Tag enthält Java-Programmcode. Das JSP-Tag steht in der Seite als Platzhalter für ein Textstück, das von dem Programmcode berechnet wird.
- x Beim Aufruf einer JSP-Seite
 - ⇒ ... führt der Server von allen in der Seite enthaltenen JSP-Tags den darin enthaltenen Programmcode aus,
 - ⇒ ... ersetzt in der JSP-Datei alle JSP-Tags durch die vom jeweiligen Programmcode berechneten Inhalte,
 - ⇒ ... sendet die so erzeugte Seite zum Client.

Festgeld-Servlet

```
public class Festgeld extends HttpServlet {  
    public void service (HttpServletRequest request, HttpServletResponse response)  
        throws IOException {  
        response.getWriter().println("<html><body><span style=\"font-family: trebuchet ms;\">");  
        response.getWriter().println("<h2>... und das kommt für Sie dabei heraus!</h2>");  
        response.getWriter().println("<p>Ihr Guthaben nach fünf Jahren: ");  
        response.getWriter().println(  
            (int) (Math.pow(1.045,5) * Integer.parseInt(request.getParameter("betrag"))) );  
        response.getWriter().println(" €</p><p> ");  
        response.getWriter().println("Diesem Angebot liegt eine Verzinsung von 4.5% zugrunde.");  
        response.getWriter().println("(Stand 20.7.2003)</p><hr/>");  
        response.getWriter().println("<p><img src=\"dreibalkenbank.jpg\" height=\"28\"/>");  
        response.getWriter().println(  
            "<span style=\"color: rgb(255, 0, 0); \"> Dreibalkenbank - Innovation braucht Stärke ");  
        response.getWriter().println("</span></p></span></body></html>");  
    }  
}
```

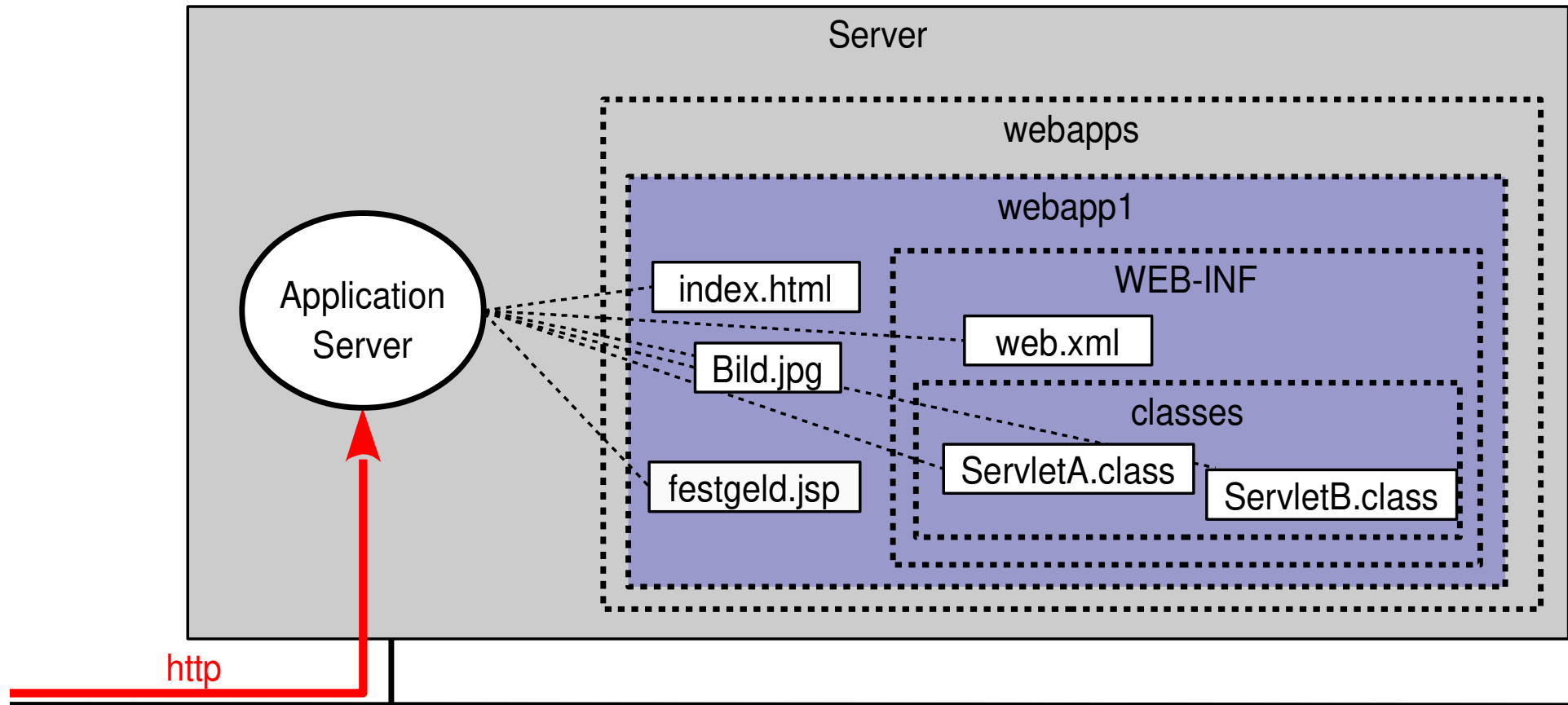
Festgeld-JSP

Datei festgeld.jsp

```
<%@ page language="java" contentType="text/html; charset=iso-8859-1" %>
<html><body><span style="font-family: trebuchet ms;">
<h2>... und das kommt für Sie dabei heraus!</h2>
<p>Ihr Guthaben nach fünf Jahren:
<%= (int)(Math.pow(1.045,5) * Integer.parseInt(request.getParameter("betrag"))) %>
€</p><p>
Diesem Angebot liegt eine Verzinsung von 4.5% zugrunde.
(Stand 20.7.2003)</p><hr/>
<p> 
<span style="color: rgb(255, 0, 0); "> Dreibalkenbank - Innovation braucht Stärke </span> </p>
</span></body></html>
```

5.12 JSPs - Grundlagen

JSPs in Web-Applications



<http://127.0.0.1:8080/webapp1/festgeld.jsp>

JSPs

- x JSPs sind Bestandteil von J2EE.
- x Alle Dateien einer Java-Web-Application, die die Endung .jsp haben, werden vom Application Server als JSP-Dateien interpretiert.
 - ⇒ Anmerkungen:
 - x JSP-Dateien befinden sich nicht im Verzeichnis WEB-INF.
 - x JSPs müssen nicht im Deployment Descriptor deklariert werden.
- x Der Client greift auf JSPs in gleicher Weise zu wie er auf andere statische Seiten zugreift.
 - ⇒ Der URL-Pfad ergibt sich aus dem Context-Path und aus dem Verzeichnis in dem die JSP-Datei innerhalb der Web-Application liegt und aus dem Namen der JSP-Datei.
- x Einziger Unterschied: Die Inhalte werden serverseitig dynamisch berechnet.

Content Type

- x Bei statischen Seiten ergibt sich der MIME-Type aus der Dateiendung.
- x Bei JSPs wird der MIME-Type explizit durch eine entsprechende Content-Type-Deklaration festgelegt.

```
<%@ page language="java" contentType="text/html; charset=iso-8859-1" %>
```


Können JSP-Seiten beliebige Datenformate beinhalten?

- x In der Content-Type-Deklaration kann ein beliebiger MIME-Type angegeben werden.
- x Die JSP-Tags sind Textstücke. Aus diesem Grund eignen sich JSPs nur für textbasierte Datenformate.

XML, HTML und JSP

- x Enthält die JSP-Datei ein Datenformat im XML-Stil, so kann JSP als eine Erweiterung dieser XML-Sprache um dynamische Tags betrachtet werden.

⇒ Beispiel:

```
<xy><b></b><z><%= request.getParameter("j") %></z></xy>
```

⇒ Nicht zulässig ist es, die `<%...%>`-Tags innerhalb anderer Tags zu platzieren. Beispiel:

```
<xy <%= request.getParameter("j") %>></xy>
```

- x Editoren, die mit Datenformaten im XML-Stil arbeiten (etwa HTML), betrachten zusätzliche JSP-Tags als Text-Inhalte oder als unbekannte Tags.
 - ⇒ Bestehende Editoren für HTML können auch sinnvoll für HTML-basierte JSPs eingesetzt werden.
 - ⇒ Es gibt auch spezialisierte JSP-Editoren mit einer spezifischen Unterstützung für JSP-, HTML- und XML-Tags.

Java-Ausdrücke in JSPs

Stellen, an denen der Inhalt der Web-Seite durch einen Java-Ausdruck dynamisch berechnet werden soll, werden mit `<%=` und `%>` geklammert.

⇒ Beispiel:

Das ist Ihre Eingabe: `<%=request.getParameter("x")%>`

Implizite Objekte

Innerhalb der JSP-Tags stehen implizit Variablen zur Verfügung, über auf verschiedene Informationen zugegriffen werden kann.

<i>Variable</i>	<i>Bedeutung</i>	<i>Typ</i>
request	Request-Objekt	HttpServletRequest
response	Response-Objekt	HttpServletResponse
session	Session-Objekt	HttpSession
application	Objekt mit Informationen zur Web-Application. Unter anderem: Zugriff auf Attribute des Application Scope.	ServletContext
pageContext	Objekt mit Daten, die sich auf die aktuelle Seite beziehen. Unter anderem: Zugriff auf Attribute des Page Scope.	PageContext

JSPs und Servlets

Der Application Server übersetzt JSPs intern in Servlets.

- x Beim Aufruf einer JSP-Seite prüft der Application Server zunächst, ob zu der JSP bereits ein Servlet erzeugt wurde. Wurde das Servlet noch nicht erzeugt, so übersetzt er die JSP-Datei zunächst in ein Servlet (Datei mit Java-Programmcode) und kompiliert dann das Servlet. Anschließend wird zur Bearbeitung des Requests das so erzeugte Servlet aufgerufen.
- x Die Erzeugung der Servlets geschieht automatisch.
- x Wenn nicht anders konfiguriert, so werden die JSPs erst dann in Servlets übersetzt, wenn es zum ersten Aufruf durch einen Client kommt. Der erste Aufruf der JSP-Seite dauert dadurch länger. Bei allen weiteren Aufrufen, existiert das Servlet bereits, der Application Server leitet den Aufruf direkt an das Servlet weiter.

Übersetzung JSP → Servlet

```
<%@ page language="java" contentType="text/html; charset=iso-8859-1" %>
<p>Ihr Guthaben nach fünf Jahren:
<%= (int)(Math.pow(1.045,5) * Integer.parseInt(request.getParameter("betrag"))) %>
€</p>
```

```
public class festgeld_jsp extends HttpServlet {
    public void service (HttpServletRequest request, HttpServletResponse response)
        throws IOException {
        response.setContentType("text/html; charset=iso-8859-1");
        response.getWriter().println("<p>Ihr Guthaben nach fünf Jahren: ");
        response.getWriter().println(
            (int) (Math.pow(1.045,5) *
                Integer.parseInt(request.getParameter("betrag"))) );
        response.getWriter().println(" €</p>");
    }
}
```

... Übersetzung JSP → Servlet

Anmerkung: Bei diesem Beispiel wurde ein wenig vereinfacht. Die konkrete Realisierung der Übersetzung kann in technischen Details anders aussehen.

Anweisungen in JSPs

- x In JSPs können beliebige Java-Anweisungen ausgeführt werden. Diese werden durch `<%` und `%>` umklammert.

⇒ Beispiel:

```
<%  
    java.util.Date d = new java.util.Date();  
    String s = "keine gerade Zeit";  
    if (d.getSeconds() % 2 == 0)  
        s = d.toString();  
%>
```

- x Die Anweisungen werden innerhalb der service-Methode des Servlets ausgeführt. Auf die hier deklarierten lokalen Variablen kann in den nachfolgenden Tags mit Java-Ausdrücken zugegriffen werden.

```
<%= s %>
```


Programmcode in JSPs

- x Mit Hilfe des Tags `<%...%>` kann beliebiger Programmcode in die Service-Methode eingefügt werden. Das folgende Beispiel erzeugt eine JSP-Fragment, mit dem eine Tabelle mit Quadratzahlen dargestellt wird.

```
<table>
<%
for (int i=0; i<5; i++) {
%>
<tr><td> <%= i%> </td><td> <%= i * i%> </td></tr>
<%
}
%>
</table>
```

Methoden und Objekt-Attribute in Servlets

```
<%!  
private int counter = 0;  
public synchronized int next () {  
    counter++;  
    return counter;  
}  
%>
```

- x Die so deklarierten Attribute und Methoden werden Teil der Servlet Klasse (neben der service-Methode). Innerhalb der Java-Ausdrücke der JSP kann auf diese zugegriffen werden.

```
<%= next() %>
```

Übersicht: Einbindung von Java-Code in JSPs

<code><%= ... %></code>	Java-Ausdruck, dessen Wert an dieser Stelle in die HTML-Seite eingefügt wird.
<code><% ... %></code>	Programmcode, der in die service-Methode eingefügt wird - zwischen die Ausgabe-Anweisungen für die statischen Textstücke und die Java-Ausdrücke <code><%= ... %></code> .
<code><%! ... %></code>	Deklaration zusätzlicher Methoden und Attribute für die Klasse Servlet. Auf diese kann aus den Tags <code><%= ... %></code> und <code><% ... %></code> zugegriffen werden.

JSPs und Multithreading

- x Ein Application Server kann eine JSP nebenläufig durch mehrere Threads gleichzeitig ausführen (analog zu Servlets).
- x Mit dem folgenden JSP-Tag kann Multithreading für diese JSP unterbunden werden.

`<%@ page isThreadSafe="false" %>`

- x Folge: Die service-Methode des zugehörigen Servlets wird immer nur von einem Thread gleichzeitig ausgeführt.
 - ⇒ Dies entspricht dem, dass die service-Methode synchronized ist.
- x Vorsicht! Während der Bearbeitung dieser service-Methode können jedoch auch gleichzeitig die service-Methoden anderer Servlets bearbeitet werden.
 - ⇒ Synchronisationsbedarf beim Zugriff auf gemeinsame Daten bleibt: Session-Attribute, Klassenattribute, etc.

Server Side Includes

Bei JSPs gibt es zwei Möglichkeiten, andere Seiten der gleichen Web-Application in die aktuelle Seite einzufügen.

- x Einfügen während die JSP in ein Servlet übersetzt wird

```
<%@ include file="banner.html" %>
```

⇒ Der Eintrag wird durch die Datei ersetzt. Anschließend wird die JSP in ein Servlet konvertiert.

- x Einfügen während der Bearbeitung des Requests

```
<jsp:include page="banner.html"/>
```

⇒ Bei der Bearbeitung eines Request ruft das Servlet an dieser Stelle eine Methode auf, die die Seite einfügt.

In beiden Fällen kann die einzufügende Seite eine beliebige Text-Datei mit einem HTML-/JSP-Teilstück sein.

Wann Servlets und wann JSPs?

- x Bei JSPs lassen sich die Inhalte leichter editieren.
 - ⇒ JSPs mit HTML-Inhalten lassen sich leicht mit HTML-WYSIWYG-Editoren bearbeiten.
 - ⇒ spezielle WYSIWYG-Editoren für JSPs
- x Bei Servlets lässt sich der Programmcode leichter warten - bei JSPs befinden sich Programmstücke verstreut in den JSP-Dateien. Empfehlungen:
 - ⇒ keine großen Programmstücke in JSPs
 - ⇒ gegebenenfalls Programmstücke zu Methoden zusammenfassen und diese dann in den JSPs aufrufen
- x JSPs werden vor der Ausführung in Servlets übersetzt. Eine JSP zu debuggen bedeutet das daraus generierte Servlet zu debuggen. Dazu muss man verstehen, wie dieser Übersetzungsvorgang funktioniert.
 - ⇒ Unterstützung bieten hier zum Teil JSP-Debugger.

Kombination Servlet-JSP

Weit verbreitet ist das nachfolgend beschriebene Schema, bei dem Servlets und JSPs kombiniert werden:

- x Vom Client aus wird das Servlet aufgerufen.
- x Das Servlet berechnet eine Menge von Daten, die auf der dynamischen Seite dargestellt werden sollen.
- x Diese Daten legt das Servlet im Request-Objekt oder im Session-Objekt ab.
 - ⇒ Request-Attribute, Session-Attribute
- x Anschließend leitet das Servlet den Request per forward an eine JSP weiter.
- x In der JSP werden lediglich die zuvor berechneten Daten an verschiedenen Stellen eingefügt. Es finden in der JSP keine weiteren Berechnungen statt.

Beispiel

```
public class FibonacciServlet extends HttpServlet {  
    public void service(HttpServletRequest request, HttpServletResponse response)  
        throws IOException {  
        int x = Integer.parseInt(request.getParameter("x"));  
        int a = 1; int b = 1; boolean isFibonacci = false;  
        while (b <= x) {  
            if (b == x)  
                isFibonacci = true;  
            int c = b + a;  
            a = b;  
            b = c;  
        }  
        request.setAttribute("fib", new Boolean(isFibonacci));  
        request.setAttribute("fibnext", new Integer(b));  
        try {  
            request.getRequestDispatcher("fibonacci.jsp").forward(request, response);  
        } catch (ServletException e) {}  
    }  
}
```


fibonacci.jsp

```
<%= request.getParameter("x") %>  
ist <%=((Boolean)request.getAttribute("fib")).booleanValue() ? "eine" : "keine" %>  
Fibonacci-Zahl.  
<p/>
```

```
Die nächste Fibonacci-Zahl ist  
<%= request.getAttribute("fibnext") %>.
```

5.13 JSP EL

Motivation

Man betrachte folgendes Teilstück einer HTML-basierten JSP:

```
") %>" border="5"/>
```

- ⇒ Es handelt sich um eine zulässige JSP-Seite.
- ⇒ Bei der Bearbeitung des Requests wird das JSP-Tag ersetzt und es entsteht eine konsistente HTML-Seite.
- ⇒ Problem: Wird diese JSP-Seite mit einem HTML-Editor bearbeitet, so führen die Zeichen im JSP-Tag dazu, dass das Ende des HTML-Tags falsch interpretiert wird.

```
") %>" border="5"/>
```

- ⇒ Allgemein: Da innerhalb der JSP-Tags alle möglichen Zeichen stehen können, sind HTML-basierte JSP-Seiten im allgemeinen nicht HTML-konform und können deshalb nicht vernünftig mit einem HTML-Editor bearbeitet werden.

... Motivation

- x JSPs können prinzipiell alle möglichen textbasierten Datenformate erzeugen. Besonders populär sind jedoch XML-basierte oder XML-ähnliche Formate - vor allem HTML.
- x Prinzipiell können JSP-Tags an beliebigen Positionen innerhalb einer JSP-Datei platziert werden.
- x Problem: Wurde die JSP-Datei so realisiert, dass sie immer konsistente XML-Seiten generieren, so heißt dies noch nicht, dass auch die JSP-Datei selbst eine konsistente XML-Datei ist und auch mit entsprechenden XML-Editoren bearbeitet werden kann.

Forderung

Benötigt werden JSP-Tags, die in XML-Dateien sowohl innerhalb von Attributwerten als auch neben Tags eingesetzt werden können. Für diese Tags sollen syntaktische Einschränkungen definiert werden, die dafür sorgen, durch das Einfügen dieser Tags in eine XML-Datei die Struktur der XML-Datei erhalten bleibt.

Folgerung:

- ⇒ Diese JSP-Tags sollen weder Anführungszeichen " noch spitze Klammern < > enthalten.

EL-Tags (Expression Language)

Beispiel:

```
  
<h1>  
  ${sessionScope.x}  
</h1>
```

- ⇒ EL-Tags werden mit \${ und } geklammert.
- ⇒ EL-Tags sollen in XML-Dateien entweder innerhalb von Attributwerten oder neben XML-Tags platziert werden.
- ⇒ EL-Tags enthalten weder doppelte Anführungszeichen noch spitze Klammern.
- ⇒ Anstelle der doppelten Anführungszeichen " werden einfache Anführungszeichen ' verwendet .

Scripting-Tags und EL-Tags

- x Man unterscheidet zwei Arten von JSP-Tags: Scripting-Tags und EL-Tags.
- x Scripting-Tags
 - ⇒ Unter Scripting-Tags versteht man die bisher erwähnten JSP-Tags. Sie enthalten Java-Programmcode und werden mit `<%` und `%>` geklammert.
- x EL-Tags
 - ⇒ EL-Tags werden mit `${` und `}` geklammert und enthalten immer einen Ausdruck der in einer eigenen Sprache, der expression language (EL), verfasst ist.
 - ⇒ EL-Tags enthalten immer nur Ausdrücke und entsprechen so dem Scripting-Tags `<%= ... %>`.

Expression Language EL

- x EL ist eine Sprache, die an Java angebunden ist. Die einzige Aufgabe von EL ist es, aus EL-Tags heraus den Zugriff auf Werte und Methoden von Java zu ermöglichen.
- x EL ist Servlet-bezogen. In EL stehen implizit mehrere Variablen zur Verfügung, die die bei der Bearbeitung eines HTTP-Request relevanten Daten repräsentieren: requestScope, sessionScope, param, initParam, ...
- x EL ist Java-ähnlich.
 - ⇒ Konstanten: 13 5.583 true false null ...
 - ⇒ Operatoren: + - == * ! && || ...
- x EL ist erweiterbar.
 - ⇒ EL kann durch entsprechende Konfigurationen um weitere Methoden aus Java angereichert werden.

EL-Variablen ohne Scope-Angabe

- x Jede Variable in EL bezieht sich auf ein Scope-Attribut in Java.
- x Die Zuordnung kann durch einen entsprechenden Scope-Prefix kenntlich gemacht werden: *applicationScope*, *sessionScope*, *requestScope*, *pageScope*
 - ⇒ Beispiel: Der Ausdruck *sessionScope.cart* steht für das Session-Attribut mit dem Namen *cart*.
- x Werden in EL Variablennamen ohne Scope-Prefix angegeben, so wird implizit eine Zuordnung zu einem Attribut in einem Scope festgelegt:
 - ⇒ Es wird nacheinander im Page-Scope, im Request-Scope, im Session-Scope und schließlich im Application-Scope nach einem Attribut mit diesem Namen gesucht.
 - ⇒ Das Attribut, das in der Suchreihenfolge zuerst gefunden wurde, wird dem Variablennamen zugeordnet.

Beispiele

<i>EL-Tag</i>	<i>entsprechender Java-Ausdruck</i>
<code>\${param.x}</code>	<code><%= request.getParameter("x") %></code>
<code>\${param['x']}</code>	<code><%= request.getParameter("x") %></code>
<code>\${requestScope.a}</code>	<code><%= request.getAttribute("a") %></code>
<code>\${param.x * 3}</code>	<code><%= Integer.parseInt(request.getParameter("x")) * 3 %></code>
<code>\${requestScope.a + 8}</code>	<code><%= ((Integer)(request.getAttribute("a"))).intValue() + 8 %></code>
<code>\${sessionScope.p ? 2 : 3}</code>	<code><%= ((Boolean)(session.getAttribute("p"))).booleanValue() ? 2 : 3 %></code>

Anmerkungen zu den Beispielen

- x In EL werden String-Konstanten durch einfache Anführungszeichen umschlossen.

⇒ Beispiel:

```

```

- x Der Umgang mit unterschiedlichen Datentypen ist pragmatisch:
 - ⇒ Wrapper-Objekte (Integer, Boolean, Double, ...) dienen der Aufbewahrung von skalaren Werten (int, boolean, double, ...). In EL können die Wrapper-Objekte wie skalare Werte verwendet werden.
 - ⇒ String-Werte, die einen skalaren Wert enthalten, können wie skalare Werte verwendet werden. Die Parse-Methode (Integer.parseInt()) wird nicht benötigt.

```
${param.x * 3}
```

Properties in Java

Man spricht in Java davon, dass eine Klasse ein Property x hat, wenn es zwei Methoden getX und setX gibt, über die sich der Wert dieses Properties ändern bzw. abfragen lässt. Die Methode getX bezeichnet man dabei als "Getter" des Properties und setX als deren "Setter".

Beispiel:

```
public class Data {  
    private int zaehler;  
    public int getZaehler() {  
        return zaehler;  
    }  
    public void setZaehler(int zaehler) {  
        this.zaehler = zaehler;  
    }  
}
```

Zugriff auf Properties mit EL

Ist k eine gültige EL-Variable und steht diese EL-Variable für ein Java Objekt mit einem Property z, so kann in EL auf dieses Property mit k.z zugegriffen werden.

Beispiel:

- ⇒ Eine Instanz der Klasse Data wird als Application-Attribut mit dem Namen "wichtig" gespeichert:

```
Data d = new Data();  
d.setZaehler(5);  
getServletContext().setAttribute("wichtig",d);
```

- ⇒ Aus der JSP heraus wird auf das Property "zaehler" des Application-Attributs "wichtig" zugegriffen.

```
Der Zähler steht jetzt auf ${applicationScope.wichtig.zaehler}.
```

EL-Tags und ältere JSP-Versionen

- x EL-Tags wurden erst in der Version 2.0 von JSP eingeführt.
⇒ ab Tomcat Version 5.0
- x Um kompatibel mit älteren Versionen zu sein, kann der Application Server durch ein entsprechendes EL-Tag in der JSP darüber informiert werden, ob EL-Tags ausgewertet oder ignoriert werden sollen.
- x Empfehlung: Sobald in einer JSP EL-Tags eingesetzt werden sollen, fügt man die folgende Zeile zu Beginn der JSP ein.

```
<%@ page isELIgnored="false" %>
```

Beispiel fibonacci.jsp

```
<%@ page isELIgnored="false" %>

<font color="${requestScope.fib ? 'red' : 'black'}">
  ${param.x}
  ist ${requestScope.fib ? 'eine' : 'keine'} Fibonacci-Zahl.
</font>
<p/>

Die nächste Fibonacci-Zahl ist
${requestScope.fibnext}.
```

... Beispiel fibonacci.jsp

Anmerkung: Anders als in der zuvor präsentierten Version von fibonacci.jsp wird hier ein zusätzliches -Tag eingesetzt, in dem der Attributwert für das Attribut color dynamisch durch ein EL-Tag berechnet wird. Dies führt hier dazu, dass der mit geklammerte Text in Abhängigkeit des Request-Parameters "fib" rot beziehungsweise schwarz dargestellt wird.

5.14 Custom Tags

Motivation

- x Der Umgang mit den bisher beschriebenen JSP-Tags erfordert Kenntnisse in der Programmiersprache Java.
- x JSPs und statische Seiten müssen in gestalterischer Hinsicht bearbeitet werden, um Ihnen ein stimmiges, einheitliches Design zu geben.
- x Gründe dafür, in einer Web-Application die "Programmieranteile" von den "Darstellungsanteilen" sauber zu trennen.
 - ⇒ Aufteilung der Aufgaben auf unterschiedliche Mitarbeiter mit unterschiedlichen Fachkompetenzen: Programmierer, Web-Designer.
 - ⇒ Bei der Bearbeitung der darstellungsrelevanten Dateien soll man sich nicht mit dem Programmcode beschäftigen müssen.
 - ⇒ Bei der Programmierung, beim Programmtest und beim Debugging soll der Programmierer nicht gezwungen sein, die Code-Fragmente an verschiedenen Stellen in Web-Seiten zu suchen.

Idee

- x Der Programmierer deklariert in so genannten Tag-Dateien neue Tags (Custom Tags) als Makros für JSP-Teilstücke.
- x Auf diese Weise wird die Sprache HTML um neue, dynamische Tags erweitert.
- x Bei der Konstruktion von JSPs verwendet der Web-Designer die erweiterte HTML-Sprache bestehend aus HTML-Tags und Custom Tags.
 - ⇒ Der Web-Designer verzichtet in den JSPs auf die Verwendung von JSP-Tags mit Java- oder EL-Code!
- x Schnittstelle zwischen Programmierer und Web-Designer: Custom Tags.

Anmerkung: Dieses Konzept ist nicht auf JSPs mit HTML-Inhalten beschränkt, sondern eignet sich ganz allgemein für XML-Datenformate. Im weiteren soll dieses Konzept jedoch allein anhand von HTML ausgeführt werden.

Beispiel

Tag-Datei: /WEB-INF/tags/**box**.tag

```
<%@ page isELIgnored="false" %>
<%@ attribute name="label" %>
<%@ attribute name="content" %>
<table>
  <tr><td bgcolor="salmon">${label}</td></tr>
  <tr><td>
    <table border="1" rules="groups"><tr><td>${content}</td></tr></table>
  </td></tr>
</table>
```

JSP-Datei

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="x"%>
Jetzt kommt die Box:<p/>
<x:box label="XY-Box" content="Die ist der Inhalt der Box."/>
```

... Beispiel

Jetzt kommt die Box:

XY-Box

Die ist der Inhalt der Box.

Erläuterungen

- x Custom Tags werden durch Tag-Dateien mit der Endung .tag deklariert. Ein Verzeichnis mit einem oder mehreren Tag-Dateien wird als eine Tag Library bezeichnet.
- x Bevor Custom Tags in einer JSP verwendet werden können, muss die Tag Library angegeben werden.

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="x"%>
```

- x Es könnte in einer Library ein Tag geben, das es mit diesem Namen bereits in HTML gibt oder das auch in einer anderen Tag Library vorkommt, die ebenfalls verwendet werden soll.
- x Aus diesem Grund wird den Tags einer Library innerhalb einer JSP ein Prefix zugeordnet, der auf die jeweilige Tag Library verweist.

```
<x:box label="XY-Box" content="Die ist der Inhalt der Box."/>
```

... Erläuterungen

- x Der Name des Custom Tags ergibt sich aus dem Dateinamen der Tag-Datei.
- x Innerhalb der Tag-Datei werden zunächst die Attribute des Tags deklariert.

```
<%@ attribute name="label" %>  
<%@ attribute name="content" %>
```

- x Unterhalb der Attribut-Deklarationen dürfen die Attribute innerhalb von EL-Tags verwendet werden.

```
<tr><td>${label}</td></tr>
```

- x Damit die EL-Tags im Rumpf nicht ignoriert werden, steht zu Anfang der Tag-Datei folgender Eintrag:

```
<%@ tag isELIgnored="false" %>
```

Der Rumpf eines Custom Tags

- x HTML-Tags können einen Rumpf umklammern, in dem sich weiterer HTML-Code befinden darf.

⇒ Beispiel:

```
<h3>Dies ist eine <b>Überschrift</b></h3>
```

- x Soll ein Custom Tag einen HTML-Rumpf umklammern, so muss in dessen Tag-Datei die Position des Rumpfs mit `<jsp:doBody/>` gekennzeichnet werden.

Beispiel

Tag-Datei: /WEB-INF/tags/box.tag

```
<%@ tag isELIgnored="false" %>
<%@ attribute name="label" %>
<table>
  <tr><td bgcolor="salmon">${label}</td></tr>
  <tr><td>
    <table border="1" rules="groups"><tr><td><jsp:doBody/>
      </td></tr></table>
  </td></tr>
</table>
```

... Beispiel - Verwendung in einer JSP-Datei

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="x"%>
<x:box label="Große Box">
    Die ist ein wenig Text. <br/> Einfach irgend etwas.
    <x:box label="Noch eine Box"> Der Inhalt der
<br/><b>zweiten</b>Box.</x:box>
    <x:box label="Letzte Box"> Der Inhalt der dritten Box. </x:box>
</x:box>
```

Optionale/obligatorische Tag-Attribute

- x In der Deklaration der Tag-Attribute kann angegeben werden, ob das Tag-Attribut bei der Verwendung des Tags angegeben werden muss (required="true") oder ob es optional ist (required="false"). Wird nichts spezifiziert, so wird angenommen, dass das Tag-Attribut optional ist.
- x Der EL-Ausdruck (empty x) prüft, ob das Attribut angegeben wurde.

```
<%@ tag isELIgnored="false" %>
<%@ attribute name="label" required="true" %>
<%@ attribute name="labelcolor" required="false" %>
<%@ attribute name="content" required="true" %>
<table>
  <tr><td bgcolor="${(empty labelcolor) ? 'salmon' : labelcolor}">${label}</td></tr>
  <tr><td>
    <table border="1" rules="groups"><tr><td>${content}</td></tr></table>
  </td></tr>
</table>
```

Custom-Tags in Tag-Dateien

- x In Tag-Dateien dürfen neben HTML-Tags auch beliebige Java-Ausdrücke und EL-Tags verwendet werden. Auf diese Weise kann auf dynamische, Servlet-bezogene Inhalte zugegriffen werden.
 - ⇒ Request-Parameter, Request-Attribute, Session-Attribute, ...
- x In Tag-Dateien dürfen auch andere Custom-Tags verwendet werden.
 - ⇒ Hierarchisches Zusammenfügen von Tags zu komplexeren Tags.

5.15 Web-Applications mit Datenbankanbindung

Datenbankanbindung bei Web-Applications

- x Durch eine geeignete Konfiguration in der Datei
META-INF/context.xml
wird der Web-Application-Server aufgefordert, ein DataSource-Objekt von
einem genau bestimmten Treiber und mit einer genau bestimmten
Konfiguration zu erzeugen.
- x Dieses Objekt stellt der Web-Application-Server der Web-Anwendung zur
Verfügung. Das Objekt erhält in der context.xml-Konfiguration einen Namen
unter diesem Namen kann die Web-Anwendung das Objekt vom Web-
Application-Server anfragen.

Beispiel für META-INF\context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/MyDB"
    auth="Container" type="javax.sql.DataSource"
    username="op" password="op"
    driverClassName="org.apache.derby.jdbc.ClientDriver"
    url="jdbc:derby://localhost:1527/op" maxActive="20" maxIdle="10"
    removeAbandoned="true" removeAbandonedTimeout="300"
    logAbandoned="true"/>
</Context>
```

Zugriff auf das DataSource-Objekt

Die Web-Anwendung kann dann im Betrieb durch einen lookup-Methodenaufruf auf das DataSource-Objekt zugreifen:

```
Context ic = new InitialContext();  
DataSource datasource =  
    (DataSource)ic.lookup ("java:/comp/env/jdbc/MyDB");
```


Erläuterungen

- x In der Konfigurationsdatei context.xml werden die Eigenschaften einer Datasource beschrieben. Dazu gehören neben den Zugriffsdaten (Datenbank-URL, User-Id, Passwort, etc.) auch der Klassenname des Datenbank-Treibers. Außerdem steht in dem Datensatz ein „JNDI-Name“. JNDI-Namen haben Pfadcharakter.
 - ⇒ Beispiel: jdbc/MyDB
- x Durch die Konfiguration eines solchen Datensatzes wird der Web-Application-Server aufgefordert, ein DataSource-Objekt zu erzeugen, dieses mit den vorgegebenen Zugriffsdaten zu versehen und dieses DataSource-Objekt der Web-Application unter dem vorgegebenen JNDI-Namen zur Verfügung zu stellen.
- x Die Web-Anwendung holt sich das DataSource-Objekt durch einen Lookup vom Web-Application-Server ab.

JNDI

Dieser Mechanismus zur Datenbankbindung basiert auf einem Java-Konzept mit dem Namen JNDI.

- ⇒ JNDI: java naming and directory interface
- ⇒ Dienst zur Speicherung von Java-Objekten in einer Baumstruktur
- ⇒ vereinfacht ausgedrückt: eine Art "Dateisystem", dessen Inhalte nicht Dateien sondern Objekte sind

Datenbanktreiber bei Web-Anwendungen

- x Damit der Web-Application-Server ein *DataSource*-Objekt für eine bestimmte Datenbank erstellen kann, benötigt der Web-Application-Server den zum Datenbanksystem passenden Datenbanktreiber.
 - ⇒ Bei Tomcat gibt es ein Verzeichnis mit dem Namen *common/libs*. Alle jar-Dateien, die dort abgelegt werden, sind automatisch auf dem Classpath des Web-Application-Servers. Alle Datenbanktreiber, die von den Web-Applications benötigt werden, sollten in dieses Verzeichnis kopiert werden
- x Der Datenbanktreiber ist dann im Web-Application-Server sichtbar, nicht aber in der Web-Application.
 - ⇒ Keine Abhängigkeit zwischen Web-Application und Datenbanktreiber

Empfehlungen

- x Da der JNDI-Lookup einen gewissen Aufwand darstellt, ist es nicht ratsam, sich das *DataSource*-Objekt vor jedem Datenbankzugriff durch einen JNDI-Lookup zu besorgen.
- x Führen Sie den JNDI-Lookup nur ein mal beim Start der Web-Application aus (in der *contextInitialized*-Methode des Context-Listeners) und speichern sie das *DataSource*-Objekt in einem Application-Scope-Attribut. Sie können dann in der gesamten Web-Application und insbesondere in JSPs via EL auf dieses Objekt zugreifen.
- x Im Anschluss an den Lookup empfiehlt es sich, das *DataSource*-Objekt zu testen. Ist die Datenbank in *context.xml* richtig konfiguriert worden? Ist die Datenbank mit diesen Zugangsdaten erreichbar? Sind die relevanten Datenstrukturen (Tabellen, Views, etc.) in der Datenbank verfügbar.

... Empfehlungen

- x Ist die Datenbank nicht verfügbar, so macht der Betrieb der Web-Anwendung keinen Sinn. In diesem Fall wirft man in der *contextInitialized*-Methode eine *RuntimeException*. Das bewirkt, dass die Web-Anwendung nicht in Betrieb genommen wird. Das Starten der Web-Anwendung scheitert, HTTP-Requests werden nicht entgegengenommen.

... Empfehlungen

```
public void contextInitialized(ServletContextEvent sce) {  
    try {  
        Context initialContext = new InitialContext();  
        Context context = (Context)initialContext.lookup("java:comp/env");  
        DataSource datasource = (DataSource) context.lookup("jdbc/DefaultDB");  
  
        try (Connection connection = datasource.getConnection()) {  
            connection.createStatement().executeQuery("select * from produkte");  
            System.out.println("Datenbank erfolgreich getestet");  
            sce.getServletContext().setAttribute("datasource",datasource);  
        }  
    } catch (Exception e) {  
        System.out.println("FEHLER: Web-App start gescheitert:\n" + e);  
        throw new RuntimeException(e);  
    }  
}
```

Änderungen bei der Datenbank-Konfiguration, Wechsel der Datenbank

- x Folgt man diesem Konstruktionsprinzip, so entstehen Web-Anwendungen, die unabhängig von konkreten Datenbanken sind und mit beliebigen – aktuellen und zukünftigen – Datenbanken betrieben werden können.
- x Um die Konfiguration zu verändern, muss kein Programmcode verändert werden, sondern lediglich die Datei *context.xml*.
- x Um von einer Datenbank zur nächsten zu wechseln, müssen lediglich die passenden Treiber in das *common/libs*-Verzeichnis kopiert werden und die *context.xml*-Datei angepasst werden.

Web-Anwendungen und Datenbanken

- x Datenbanksysteme erlauben in der Regel den Betrieb mehrerer von einander unabhängiger Datenbanken.
 - ⇒ Jede Datenbank enthält eine eigene Datenbankstruktur bestehend aus Tabellen, Sichten, etc.
 - ⇒ Für jede Datenbank kann eine gesonderte Authentifizierung und Autorisierung festgelegt werden.
- x Der Name der Datenbank ist Bestandteil des Datenbank-URLs. Der folgende URL verweist auf eine Datenbank mit dem Namen **MyDB**, die sich auf einem MySql-System eines Servers mit dem Namen *server-xy.de* befindet.

jdbc:mysql://server-xy.de/**MyDB**

Empfehlungen

- x für jede Web-Application eine eigene Datenbank
- x die Datenbank erhält den gleichen Namen wie die Web-Application
- x für jede Web-Application eine eigene Datenbank-User-Id mit Lese- und Schreibrechten für die Tabellen der Datenbank
- x die User-ID erhält den gleichen Namen wie die Web-Application

Authentifizierung, benutzerbezogene Daten

- x Eine Web-Application kann so realisiert werden, dass bestimmte Bereiche der Web-Application vom Benutzer erst nach einer erfolgreichen Authentifizierung zugreifbar sind.
- x In diesem Bereich der Web-Application sind alle Clients authentifiziert und über die Methode *getRemoteUser()* kann die Web-Application die User-Id abfragen, um dann darauf aufbauend Daten zu verarbeiten, die dem Benutzer zugeordnet sind.
- x Empfehlung:
 - ⇒ Benutzerbezogene Daten können in der Datenbank durch Tabellen mit einer zusätzlichen Spalte "Benutzer-Id" realisiert werden. Jeder Wert in dieser Spalte enthält eine Web-Application-User-Id und ordnet diesen Eintrag (Zeile) so einem Benutzer der Web-Application zu.

Kapitel 6

REST-Services

REST-Services, REST-APIs

- x REST = Representational State Transfer
- x Dieser Begriff steht allgemein für ein Paradigma in der Softwarearchitektur.
- x Im engeren Sinne – und so wird er in der Regel verwendet – bezieht sich der Begriff auf Webservices, die nach diesem Funktionsprinzip arbeiten. Diese Dienste sollen als REST-Services oder REST-APIs bezeichnet werden.
- x REST-Services dienen dem Austausch von Daten zwischen Computerprogrammen.
 - ⇒ Abgrenzung: Gewöhnliche Webportale auf der Basis von HTTP dienen zur Mensch-Maschine-Kommunikation. REST-APIs dienen der Kommunikation zwischen Programmteilen, die auch auf unterschiedlichen Computern laufen können (Maschine-Maschine-Kommunikation)

Motivation REST-API

- x Unter einer API (Application Programm Interface) versteht man eine Programmierschnittstelle. Oft wird dieser Begriff gleichgesetzt mit dem Begriff Library – einer Sammlung von Klassen.
- x Unter einer REST-API versteht man eine Programmier-Schnittstelle, bei der die Funktionen der API nicht über gewöhnliche Methodenaufrufe, sondern über HTTP-Requests aufgerufen werden.
 - ⇒ Der Name der Funktion und die Parameterwerte werden mit der Request vom Client zum Service übertragen.
 - ⇒ Auf dem Service wird dann ein Stück Programmcode (Methode) ausgeführt. Das Ergebnis der Berechnung ist ein Rückgabewert bzw. eine Exception.
 - ⇒ Rückgabewert bzw. Exception werden mit der Response zurück an den Client übermittelt.

Aufbau von Request und Response

- x Es gibt verschiedene Möglichkeiten wie in dem Request und Response aufgebaut sind.
 - ⇒ Requests müssen den Funktionsnamen und die Parameterwerte enthalten
 - ⇒ Responses müssen den Rückgabewert bzw. das Exception-Objekt enthalten.
 - ⇒ Das REST-Paradigma legt nicht fest, wie diese Informationen kodiert werden.
- x Im einfachsten Fall verwendet man den Nutzdatenteil des Requests zur Übertragung des Funktionsnamen und der Parameterwerte und den Nutzdatenteil der Response für die Rückgabewerte bzw. für die Exception.
- x Es gibt unterschiedliche Möglichkeiten, wie diese Daten im Request und in der Response kodiert werden können.
 - ⇒ Gebräuchliche Kodierungen sind u.a.: JSON, XML, reiner Text, CSV.
 - ⇒ Das JSON-Format ist sehr verbreitet.

... Aufbau von Request und Response

x Alternativen

- ⇒ Der Funktionsname kann als Teilstück des Pfads des URLs übertragen werden.
- ⇒ Alternativ dazu kann der Funktionsname auch im Rumpf des Requests enthalten sein.
- ⇒ Alternativ oder auch ergänzend dazu kann die HTTP-Methode dafür verwendet werden die Funktion festzulegen.
- ⇒ Der HTTP-Statuscode in der Response kann dazu verwendet werden zwischen einer regulären Ausführung mit Rückgabewert und einer Exception zu unterscheiden. Im Fall einer erfolgreichen Ausführung wird üblicherweise der HTTP-Statuscode 200 verwendet, im Fall von Exceptions wird üblicherweise eine der Nummern 5xx verwendet. Die führende 5 steht für einen serverinternen Fehler.

CRUD-Operationen als REST-Funktionen

Möchte man mit Hilfe von REST-Funktionen Operationen auf elementaren Datensätzen durchführen, dann wird oft so verfahren, dass man den anzusprechenden Datensatz mit dem URL beschreibt und die Operation mit der HTTP-Methode.

CRUD	HTTP-Methode
create	POST
read	GET
update	PUT
delete	DELETE

Serialisierung, Deserialisierung

- x In einer Programmiersprache werden Daten in Variablen gespeichert. Bei der Durchführung einer Rest-Funktion müssen Parameterwert, Rückgabewerte und Exceptions in geeigneter Kodierung übertragen werden.
- x Serialisierung
 - ⇒ Der Variablenwert wird kodiert.
- x Deserialisierung
 - ⇒ Der Variablenwert wird dekodiert.

Anwendungsgebiete für REST-APIs

- x REST-API innerhalb von AJAX-basierten Webportalen.
 - ⇒ die REST-API als interner Teil des Web-Portals.
- x Allgemeine Informationsanbieter
 - ⇒ openweathermap: Wetterdaten weltweit
 - ⇒ omdb: Open Movie Database
 - ⇒ newsapi: Aktuelle Nachrichten weltweit
 - ⇒ ...
- x REST-APIs zur Beobachtung und Steuerung von Geräten (Internet of Things - IoT)
 - ⇒ Rasenmäher
 - ⇒ Heizung
 - ⇒ ...

Anwendungsgebiete für REST-APIs

x REST-APIs zu Backend-Systemen

⇒ Datenbanken

⇒ File Transfer

⇒ Mail

⇒ ...

Clients für REST-APIs

- x REST-APIs können aus allen möglichen Computerprogrammen heraus verwendet werden: Web-Portale, Desktop-Programme, Apps für Smartphones, Dienste jeglicher Art.
- x REST-APIs können frei zugänglich sein. Man jedoch auch festlegen, dass REST-APIs erst nach einer Authentifizierung genutzt werden können.
 - ⇒ Zugriff auf personenbezogene Daten
 - ⇒ bezahlte Dienstleistungen
- x In diesem Fall hat die Web-Anwendung, die die REST-API betreibt, eine Benutzerverwaltung. Die Authentifizierung erfolgt über das HTTP-Protokoll.

Authentifizierung und Autorisierung

- x Zur Authentifizierung kann die Basic Authentication verwendet werden.
 - ⇒ Mit jedem Aufruf übergibt der Client in einer Header-Zeile Benutzername und Passwort.
- x Einige REST-APIs verwenden raffiniertere Mechanismen zur Authentifizierung und Autorisierung. Mit Hilfe dieser Mechanismen wird es möglich,
 - ⇒ dass sich der Anwender direkt bei der REST-API anmelden kann und dass die Anwendung, die er zum Zugriff auf die REST-API verwendet, die Anmeldedaten für die REST-API nicht erfährt
 - ⇒ dass sich nur zuvor auf der REST-API registrierte Anwendungen mit der REST-API verbinden können
- x Beispiele:
 - x OAUTH2 - Open Authentication 2.0
 - x OpenID Connect
 - x JSON Web Token
 - x Spring Security
 - x ...

6.1 JSON

JSON

- x JSON: JavaScript Object Notation
- x Kodierung von Datenstrukturen in Textform
- x Ursprünglich nur im Kontext von Javascript verwendet, wird JSON inzwischen in unterschiedlichen Programmiersprachen und Anwendungsbereichen eingesetzt.

Aufbau eines JSON-Textes

x Elementare JSON-Werte sind:

- ⇒ Zahlenwerte, ganzzahlig oder Fließkommazahlen
 - ⇒ Beispiele: 4 -13.85 4.17e17
- ⇒ Boolesch Werte: true oder false
- ⇒ Zeichenketten
 - ⇒ Beispiele: "Haus" "Maus" "2021-05-16 07:35"
- ⇒ der Nullwert null

x JSON-Objekte

- ⇒ Eine Menge von Name-Wert-Paaren. Namen werden in Doppelten Anführungszeichen geschrieben. Name und Wert werden durch einen Doppelpunkt : voneinander getrennt. Das JSON-Objekt beginnt mit einer öffnenden Mengenklammer { , es folgt eine Liste von Name-Wertpaaren, die untereinander mit Kommata getrennt werden und schließlich eine schließende Mengenklammer } .
 - ⇒ Beispiel: { "x" : 5 , "y" : 7 }

... Aufbau eines JSON-Textes

x JSON-Arrays

⇒ Eine endliche Folge von Werten. Das JSON-Array beginnt mit einer eckigen Klammer [und endet mit einer eckigen Klammer] . Dazwischen stehen mehrere JSON-Werte mit Kommata voneinander getrennt.

⇒ Beispiel: [7, 2, 8]

- x Ein JSON-Text ist entweder ein elementarer JSON-Wert, ein JSON-Objekt oder ein JSON-Array.
- x Sowohl JSON-Objekte und JSON-Arrays enthalten andere JSON-Elemente, diese Elemente können entweder elementare JSON-Werte oder auch selbst wiederum JSON-Objekte oder JSON-Arrays sein.
- x Ein JSON-Array ist eine endliche Folge von Werten. JSON legt nicht fest, dass alle Elemente dieser Folge die gleiche Struktur haben (“den gleichen Typ“).

Beispiel

```
{  
  "Vorname" : "Luise",  
  "Nachname" : "Zimmermann",  
  "AusgelieheneBücher" : 17,  
  "Geburtsdatum" : "2021-02-06",  
  "Hobbys" : [ "Klettern", "Minigolf", "Klavier" ],  
  "Führerschein" : {  
    "Klassen" : [ "AM", "B", "BE" ],  
    "Ausstellungsdatum" : "2021-10-04"  
  }  
}
```

JSON und Javascript

- x In Javascript ist JSON tief verankert.
 - ⇒ der JSON-Array entspricht dem Javascript-Array
 - ⇒ das JSON-Objekt entspricht dem Typ Object in Javascript
 - ⇒ Javascript-Objektstrukturen können in Javascript direkt als Werte in JSON angegeben werden (Konstruktoraufrufe).
 - ⇒ Serialisierung mit der Funktion *JSON.stringify*
 - ⇒ Deserialisierung mit der Funktion *JSON.parse*
- x Javascript hat keine strenge Typisierung, sondern eine dynamische Typisierung. Datentypen ergeben sich zur Laufzeit nicht zur Compilierzeit.
 - ⇒ Wird ein Stück Javascript-Programmcode mehrfach durchlaufen, so können die darin enthaltenen Variablen bei jedem Durchlauf einen anderen Typ haben.
 - ⇒ Die Elemente eines Javascript-Arrays müssen nicht alle den gleichen Typ haben.

JSON und Javascript

- x Nicht jeder elementare Datentyp von Javascript entspricht einem Datentyp von JSON. Beispiele:
 - ⇒ JSON-Zahlen sind entweder int-Werte oder double-Werte
 - ⇒ Beispielsweise gibt es den Javascript-Datentyp *Date* nicht in JSON.
 - x Bei der Serialisierung mit *Json.stringify* wird ein *Date*-Wert in einen String umgewandelt.
 - ⇒ Verschiedene Javascript-Datentypen sind nicht serialisierbar:
 - x Funktionen (Lambdas)
 - x Streams
 - x Threads
 - x ...

Wertangabe in JSON-Notation

```
const person = {  
  name : "Hans Mustermann",  
  körpergröße : 182.5,  
  haustiere : [  
    { tierart: "Katze", gewicht: 4.2},  
    { tierart: "Hund", gewicht: 14}  
  ],  
  hobbies : [ "Schach", "Kite-Surfen", "Kreuzworträtsel" ]  
}
```

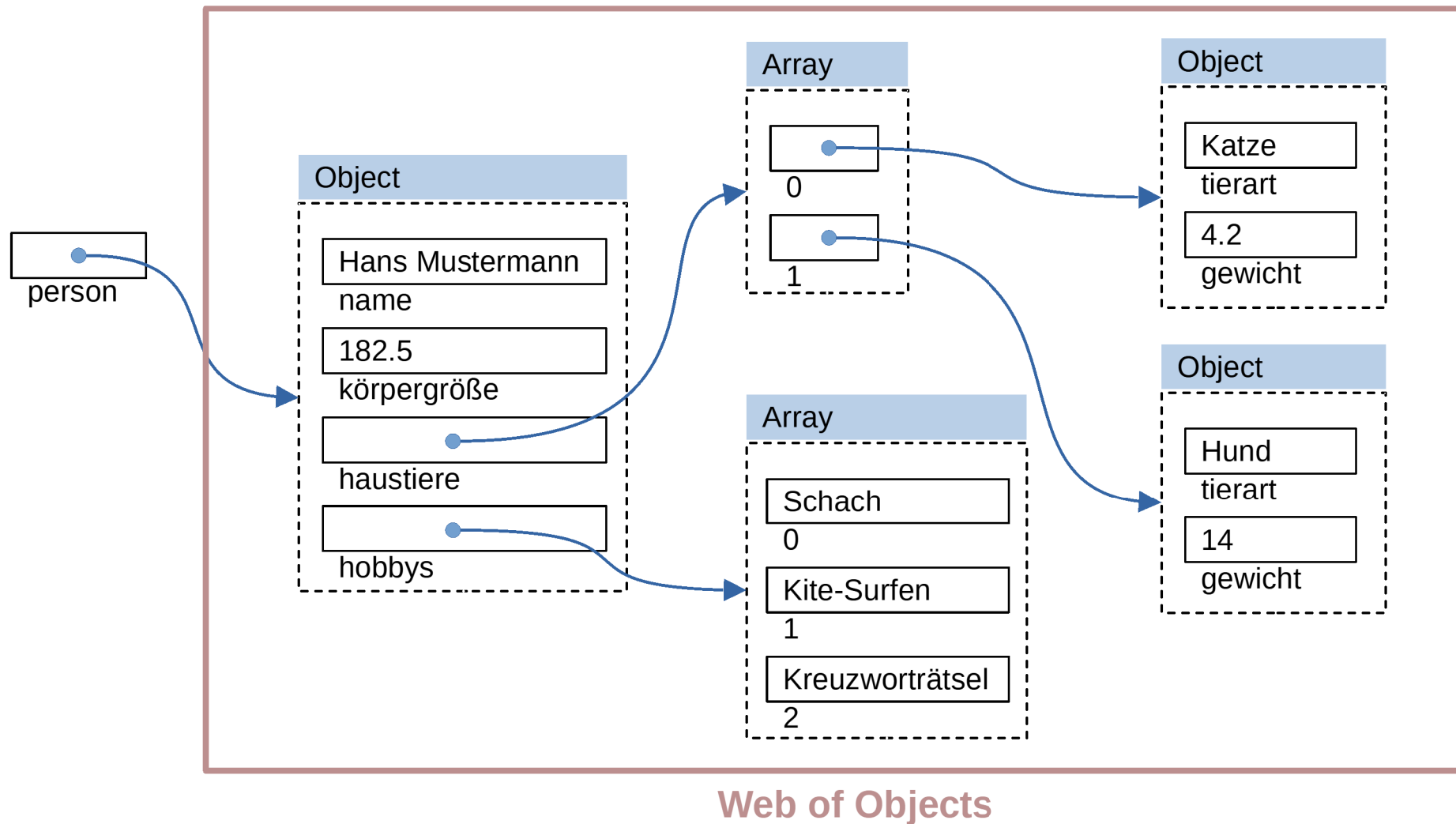
Verweise, Web of Objects

- x Variablen vom Javascript-Typ Object sind Verweise. Gleiches gilt in Javascript für Arrays.
- x Ist in Javascript der Wert einer Variablen ein Objekt oder ein Array, so können die darin enthaltenen Werte wiederum auf andere Objekte und Arrays verweisen. Das Netzwerk aus Objekten und Arrays, auf die man von einer Variablen aus direkt oder indirekt zugreifen kann, bezeichnet man als Web-of-Objects.
- x Bei der Serialisierung muss das gesamte Web of Objects serialisiert werden.

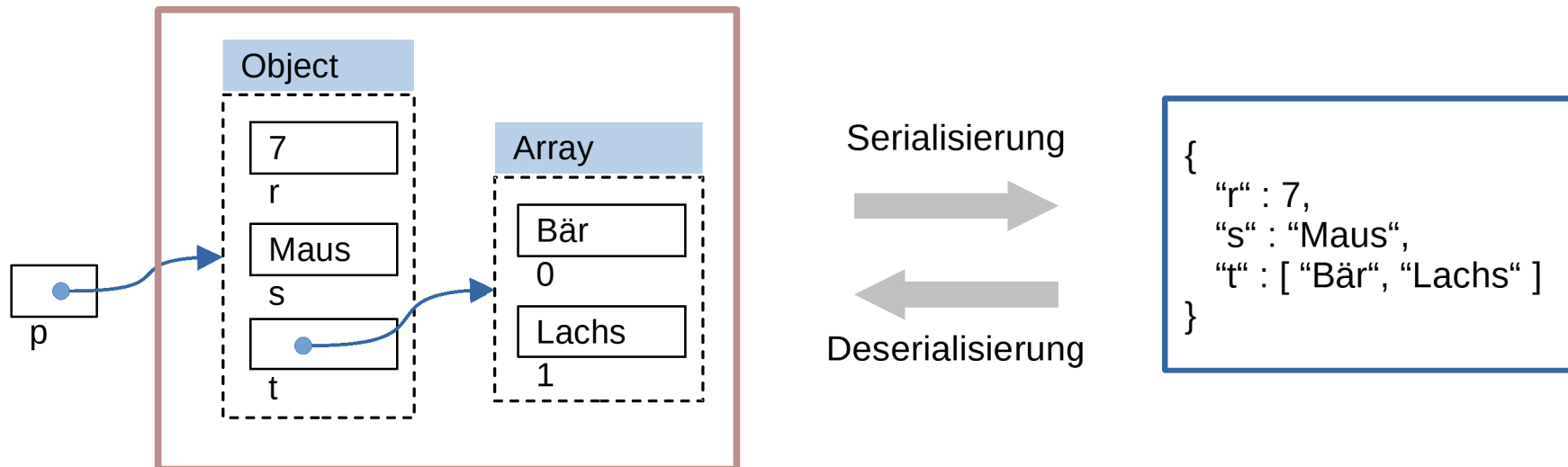
Web of Objects vs. JSON

- x Ein JSON-Text repräsentiert eine Baumstruktur. In einer Baumstruktur gibt es keine Zyklen und auch nicht mehrere Verweise auf das selbe Element.
- x Datenstrukturen in Javascript hingegen sind nicht unbedingt Bäume.
- x Verweisen innerhalb des Web of Objects mehrere Verweise auf das selbe Object/Array, so führt die Serialisierung und die anschließende Deserialisierung dazu, dass nicht mehr die selbe Struktur entsteht wie zuvor. Das neue Web of Objects enthält nicht mehr zwei Verweise auf das selbe Object/Array, sondern zwei Verweise auf zwei Objects/Arrays gleichen Inhalts.
- x Die Serialisierung scheitert, wenn das Web of Objects Zyklen enthält.

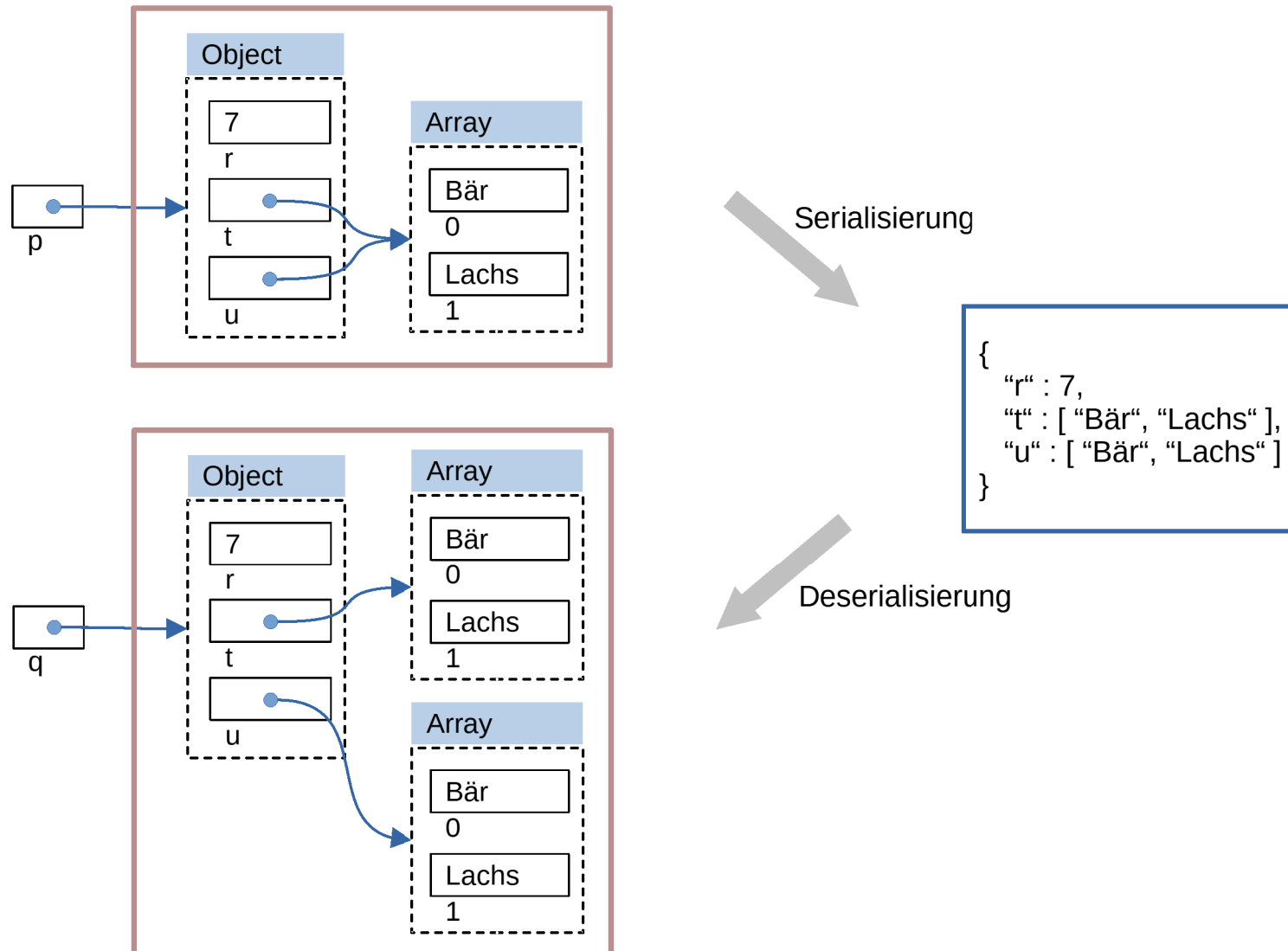
Das Objekt und sein Web of Objects



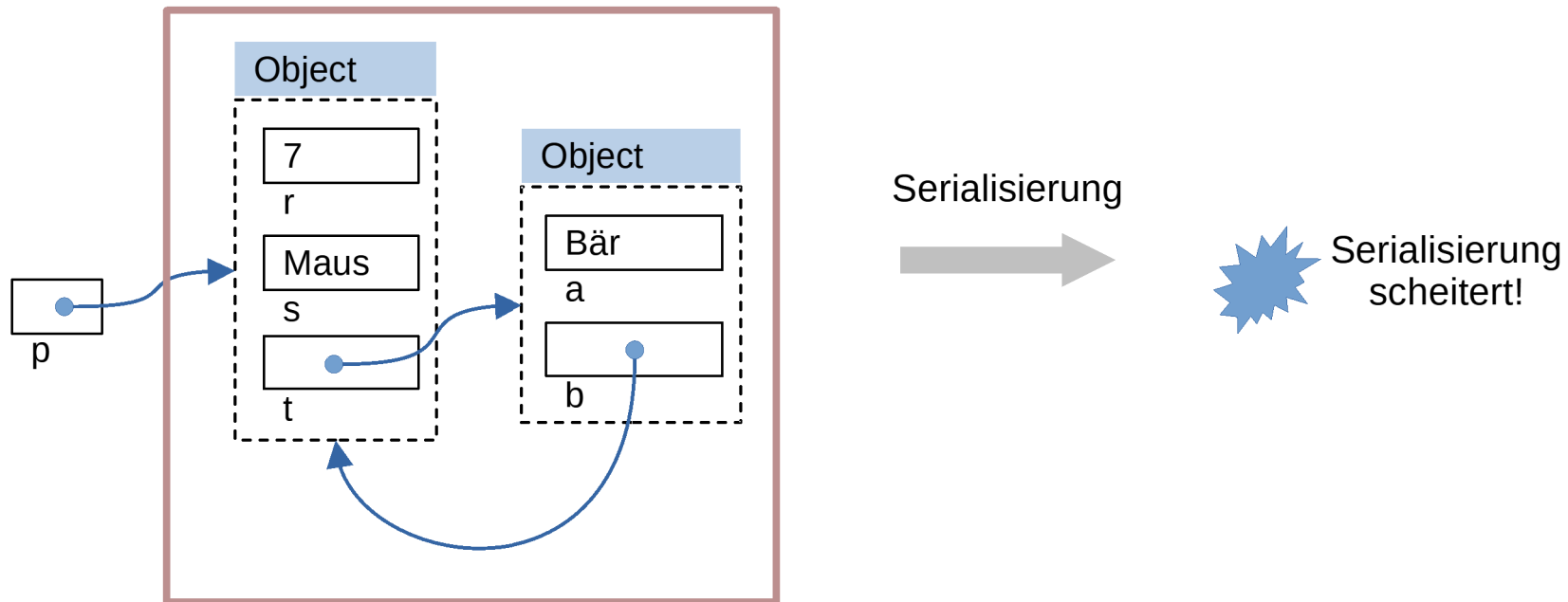
Serialisierung, Deserialisierung



Serialisierung, Deserialisierung



Serialisierung, Deserialisierung



6.2 Werkzeuge zur Serialisierung und Deserialisierung

Werkzeuge zur Serialisierung und Deserialisierung

- x Prinzipiell kann die Serialisierung und Deserialisierung von Datenstrukturen in allen gängigen Programmiersprachen selbst programmiert werden.
- x Alle gängigen Programmiersprachen enthalten Werkzeuge zur Serialisierung und Deserialisierung in JSON.
 - ⇒ Entsprechende Werkzeuge gibt es auch für alternative Kodierungen: SOAP, XML-RPC, IIOP
- x Die verschiedenen Werkzeuge verwenden unterschiedliche Konzepte und sind für unterschiedliche Anwendungsfälle geeignet auszuwählen.

Serialisierung und Deserialisierung in Java

In Java gibt es verschiedene Werkzeuge zur Serialisierung und Deserialisierung mit unterschiedlichen Datenformaten.

- x JsonObjectBuilder (JSON, nur Serialisierung)
- x Jackson (JSON)
- x GSON (JSON)
- x JAXB (XML)
- x Xstream (XML)
- x ObjectOutputStream, ObjectInputStream (Binärformate: iiop, rmi)
- x Kryo (proprietäres Binärformat)
- x ...

JsonObjectBuilder

- x Der *JsonObjectBuilder* ist Bestandteil der Java Standard Library.
- x Mit dem *JsonObjectBuilder* wird in Java eine baumförmige Objektstruktur aufgebaut. Sobald die Baumstruktur aufgebaut ist, kann sie in einen JSON-String umgewandelt werden.
- x Ausgangspunkt ist ein Objekt vom *JsonBuilderFactory*. Damit können die *Array*- und *Object*-Elemente erzeugt werden. Mit *add*-Operationen können in die *Array*- und *Object*-Elemente Daten eingefügt werden – auch Verweise auf andere *Array*- und *Object*-Elemente.
- x Der *JsonObjectBuilder* enthält nur Mechanismen für die Serialisierung, nicht aber für die Deserialisierung.

JsonObjectBuilder

```
public class ChartTable1D {  
  
    public String[] labels;  
    public double[] values;  
  
    private static JsonObjectBuilderFactory factory = JsonObjectBuilderFactory.create(new HashMap<String, Object>());  
  
    public String toChartJSData() throws JsonProcessingException {  
        JsonArrayBuilder labelsJson = factory.createArrayBuilder();  
        for (String l : labels)  
            labelsJson.add(l);  
        JsonArrayBuilder datasetsJson = factory.createArrayBuilder();  
        JsonObjectBuilder datasetJson = factory.createObjectBuilder();  
        JsonArrayBuilder vJson = factory.createArrayBuilder();  
        for (double x : values)  
            vJson.add(x);  
        datasetJson.add("data", vJson);  
        datasetsJson.add(datasetJson);  
        JsonObjectBuilder json = factory.createObjectBuilder();  
        json.add("labels", labelsJson);  
        json.add("datasets", datasetsJson);  
        return json.build().toString();  
    }  
}
```


Jackson

- x Jackson ist eine Java-Library, mit deren Hilfe Java-Objekte serialisiert und deserialisiert werden können. Jackson ist nicht Bestandteil der Standard-Library.
- x Ausgangspunkt aller Operationen ist bei Jackson ein Objekt der Klasse *ObjectMapper*.

```
ObjectMapper objectMapper = new ObjectMapper();
```

- x Die Jackson-Klasse *JsonNode* steht für eine Baumstruktur. Es gibt zwei Möglichkeiten zur Serialisierung/Deserialisierung:
 - ⇒ Man verwendet in Java *JsonNode*-Objekte.
 - ⇒ Man verwendet in Java beliebige Datentypen.

Jackson: Serialisierung und Deserialisierung mit JsonNode-Objekten

- x Jedes *JsonNode*-Objekt enthält eine Menge von Name/Wert-Paaren (Hashtable) sowie eine Menge von Verweisen auf Sohn-Knoten. Mehrere miteinander verbundene *JsonNode*-Objekte bilden eine Baumstruktur, wie man sie in JSON vorfindet.

- x Serialisierung mit *JsonNode*-Objekten:

```
String s = objectMapper.writeValueAsString(x);
```

- ⇒ Der Wert *x* ist eine Baumstruktur vom Typ *JsonNode*. Durch diesen Befehl wird *x* serialisiert und in einen JSON-String *s* überführt.

- x Deserialisierung mit *JsonNode*-Objekten:

```
JsonNode x = objectMapper.readTree(s);
```

- ⇒ Der JSON-String *s* wird deserialisiert und in eine Baumstruktur vom Typ *JsonNode* überführt.

Jackson: Serialisierung und Deserialisierung mit beliebigen Java-Objekten

- x Ein JSON-String enthält keine eindeutige Datentypen. Java ist streng typisiert. Beim Serialisieren und Deserialisieren von JSON in beliebige Java-Datentypen gibt es verschiedene Möglichkeiten, das Serialisieren und Deserialisieren zu steuern und dabei festzulegen, auf welche Java-Datentypen welche JSON-Elemente abzubilden sind.
- x Für das Serialisieren und Deserialisieren in „beliebige“ Java-Datentypen gibt es Einschränkungen. Ganz so beliebig kann der Datentyp nicht sein.
 - ⇒ Es gelten ähnliche Einschränkungen wie in Javascript (Zyklen, nicht serialisierbare Datentypen).
 - ⇒ Java ist streng typisiert. Das führt zu weiteren Einschränkungen. Beispielsweise können JSON-Arrays mit Werten von unterschiedlichen Typen können nicht in einen Java-Array konvertiert werden.

Jackson: Serialisierung von beliebigen Java-Objekten

```
String s = objectMapper.writeValueAsString(x);
```

- x* Der Wert *x* ist ein beliebiges Java-Objekt. Durch diesen Befehl wird *x* serialisiert und in einen JSON-String *s* überführt.
- x* In Jackson kann mit Hilfe von verschiedenen Annotations (Metadata, @XY) gesteuert werden, wie die Serialisierung genau erfolgen soll.

Jackson: Deserialisierung von beliebigen Java-Objekten

```
X x = objectMapper.readValue(s, X.class);
```

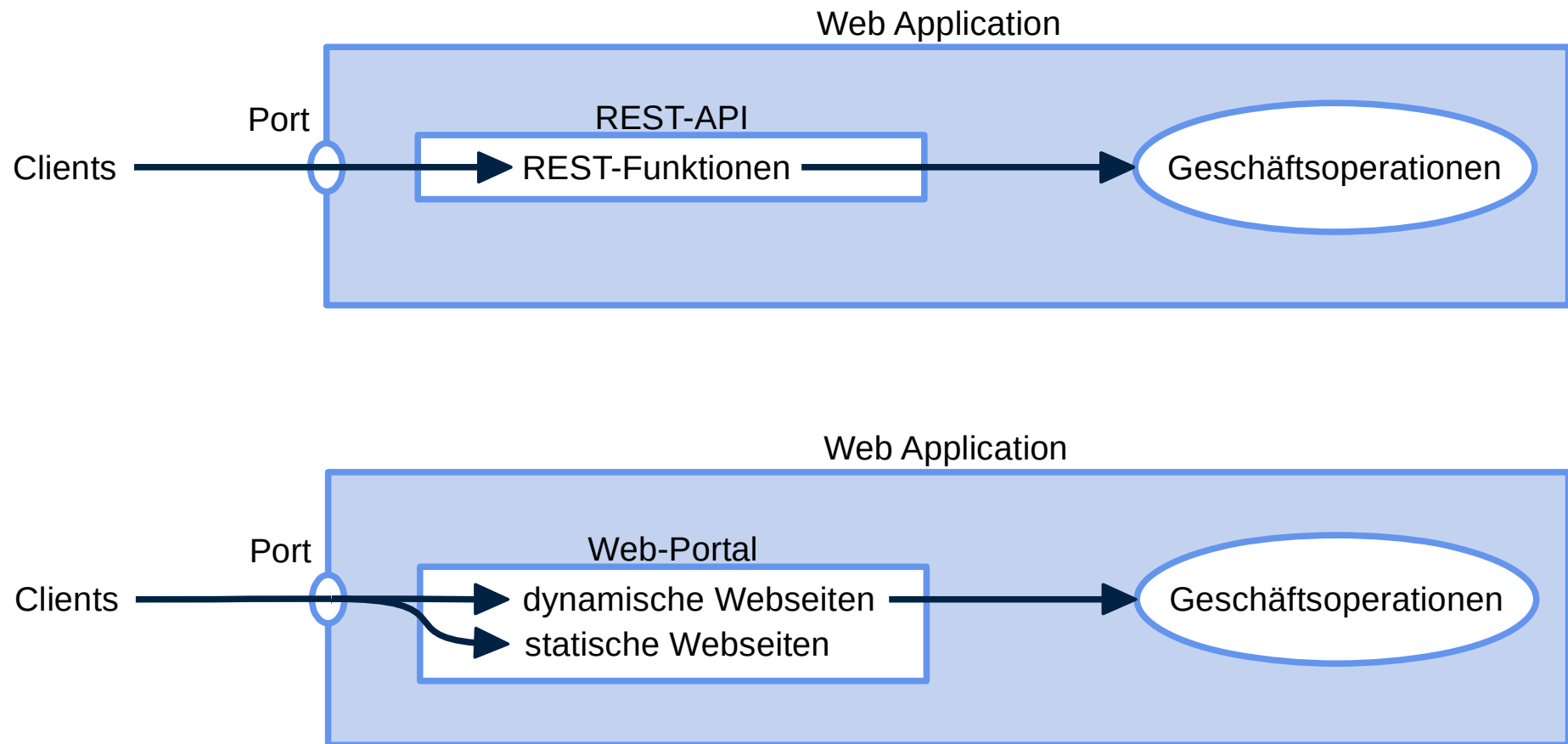
- x Der JSON-String *s* wird deserialisiert. Bei der Deserialisierung muss angegeben werden, von welcher Klasse das Objekt sein soll, das bei der Deserialisierung erzeugt werden soll. Es sei *X* eine beliebige Java-Klasse. Der Parameterwert *X.class* enthält Strukturinformationen zu der Klasse *X*. Die Methode *readValue* ist eine generische Methode (Typvariable) und erzeugt eine Instanz der Klasse *X*.
- x Es kann sein, dass der JSON-String Daten enthält, die nicht zu der Klasse *X* passen. Dann scheitert die Deserialisierung.
- x Es kann sein, dass die zu deserialisierende Instanz der Klasse *X* Objektattribute mit Verweisen auf andere Objekte enthält. Das gesamte Web of Objects muss deserialisiert werden.

6.3 Konstruktion von REST-Services

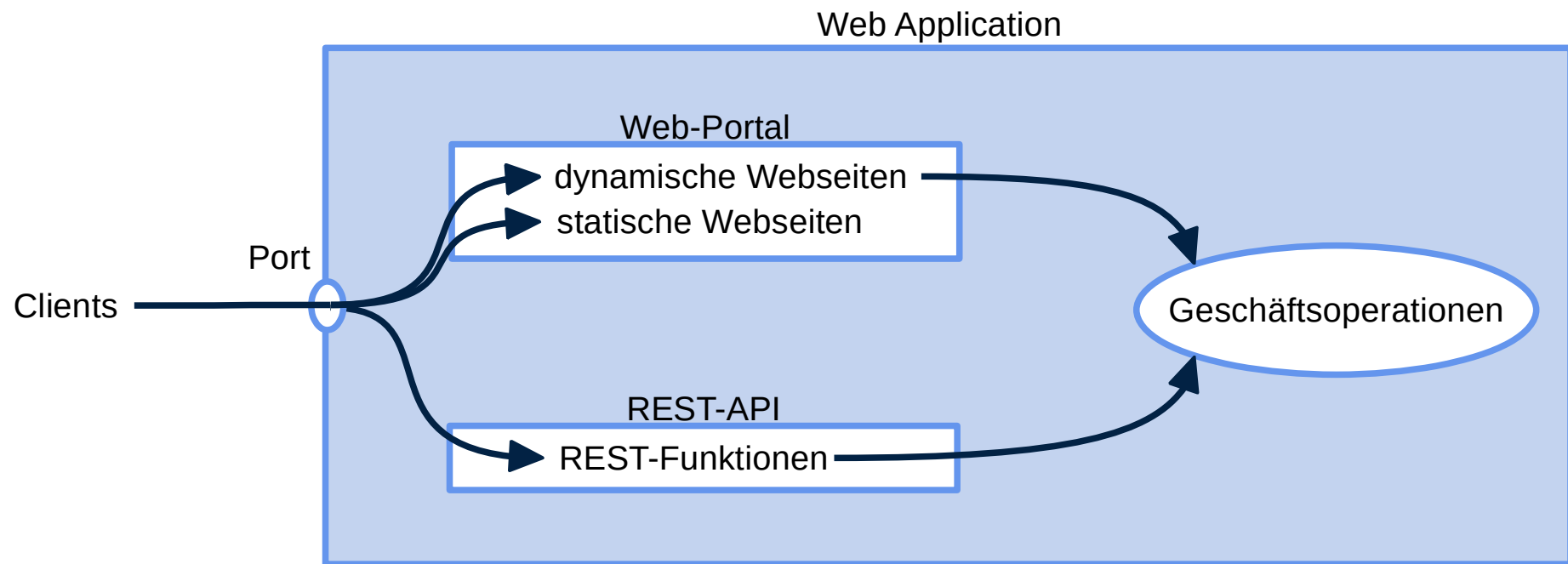
Web Applications und REST-Services

- x In der Regel verwendet man zur Konstruktion von REST-Services einen Web Application Server.
- x Klassische Web-Anwendung
 - ⇒ Die statischen und dynamische Webseiten einer klassischen Web-Anwendung erzeugen Inhalte, die dem Anwender angezeigt werden sollen. Oft erzeugen dynamische Webseiten HTML-Code.
- x Web-Anwendung als REST-API
 - ⇒ Die Web-Anwendung enthält nur dynamische Webseiten. Die dynamischen Webseiten implementieren die REST-Funktionen. Sie verarbeiten die Funktionsaufrufe.
- x Auch Mischformen sind weit verbreitet. Die Web-Anwendung stellt in diesem Fall sowohl eine Schnittstelle für den Anwender bereit, als auch REST-Funktionen.
- x Oft sind klassische Web-Anwendungen so gebaut, dass sie im laufenden Betrieb auf eine REST-API zugreifen – auf die REST-API, die in der selben Web-Anwendung enthalten ist, oder auch auf externe REST-APIs.

Web Applications und REST-Services



Web Applications und REST-Services



REST-API mit Java, Jersey

- x Prinzipiell können REST-Funktionen in einem beliebigen Application-Server als dynamische Seiten in einer beliebigen Programmiersprache implementiert werden.
- x In Java können REST-Funktionen als Java-Servlets implementiert werden.
- x Gegeben eine Java-Methode. Wie macht man aus der Java-Methode eine REST-Funktion?
 - ⇒ Teilaufgaben:
 - ⇒ entgegennehmen der Methodenparameter aus dem Request
 - ⇒ Ausführen der Methode
 - ⇒ Rückgabewert / Exception in der Response kodieren
- x Mit dem Java-Framework Jersey können in einfacher Weise aus Java-Methoden REST-Funktionen gemacht werden.
 - ⇒ Jersey ist die Referenzimplementierung des JAX-RS Standard (Jakarta RESTful Web Services).

Jersey Konfiguration

- x Die folgenden Einträge im Deployment-Descriptor *WEB-INF/web.xml* legen fest, dass Jersey in der Webanwendung unter dem Pfad */rest* einen REST-Service betreiben soll.

```
<servlet>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
</servlet>
<servlet-mapping>
  <servlet-name>javax.ws.rs.core.Application</servlet-name>
  <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

- x Soll eine Methode einer beliebigen Klasse als REST-Funktion verwendet werden können, so müssen die Klasse und die Methode mit entsprechenden Annotations (Metadata) gekennzeichnet werden.
- x Das Jersey-Framework durchsucht die Java-Klassen nach derartigen Methoden und macht aus ihnen REST-Funktionen.

Beispiel REST-Funktionen

```
@Singleton
@Path("/shop")
public class Shop {

    ...

    @Path("/products-of-category")
    @GET
    @Produces({ APPLICATION_JSON })
    public Vector<Product> getProductsOfCategory(@QueryParam("category") int category)
        throws SQLException {
        Vector<Product> products = new Vector<Product>();
        try (Connection connection = datasource.getConnection()) {
            PreparedStatement ps = connection.prepareStatement(
                "SELECT * FROM products where category = ?");
            ps.setInt(1, category);
            ResultSet rs = ps.executeQuery();
            while (rs.next())
                products.add(new Product(rs.getInt("id"), rs.getString("name"), rs.getDouble("price"),
                    rs.getInt("stock"), getCategory(rs.getInt("category"))));
        }
        return products;
    }
}
```

Beispiel-Aufruf der REST-Funktion

<http://127.0.0.1:8080/rest/shop/products-of-category?category=1>

```
[
  {
    "id": 1,
    "name": "Ball",
    "price": 29.9,
    "stock": 22,
    "category": {
      "id": 1,
      "name": "Bekleidung"
    }
  },
  {
    "id": 3,
    "name": "Lederschuhe, Damen",
    "price": 79,
    "stock": 38,
    "category": {
      "id": 1,
      "name": "Bekleidung"
    }
  }
]
```

Annotations

- x Annotations sind Java-Sprachelemente. Sie beginnen mit dem Symbol @. Annotations enthalten Konfigurationsinformationen.
- x Annotations beziehen sich immer auf das Sprachkonstrukt, vor dem sie unmittelbar stehen. Annotations können vor Klassen, Methoden, Konstruktoren, Attributen und Parametern stehen.
- x Durch das Einfügen der Annotations in den Programmcode ändert sich das Verhalten der Klasse nicht. Frameworks können jedoch die Informationen von außen auslesen.
- x Das Framework Jersey verwendet beispielsweise diese Informationen. Das Jersey-Framework durchläuft alle Klassen und erkennt bei der Klasse *Shop* an den Annotations, dass die Klasse *Shop* REST-Funktionen enthält, die das Jersey-Framework betreiben soll.
- x Viele moderne Programmiersprachen haben Annotations. Oft werden Annotations auch als Metadata bezeichnet.

Annotations in der Klasse Shop

@Singleton

Von der Klasse Shop soll eine Instanz erstellt werden. Alle Aufrufe von REST-Aufrufen sollen an diese eine Instanz geleitet werden.

@Path("/shop")

Die REST-Funktionen der Klasse Shop sollen über den URL-Pfad /shop angesprochen werden

@Path("/products-of-category")

Die Methode *getProductsOfCategory* soll über den Pfad /products-of-category als eine REST-Funktion aufgerufen werden können.

@GET

Diese REST-Funktion soll mit einem HTTP-GET aufgerufen werden.

@QueryParam("category")

Der Methodenparameter *category* soll der REST-Funktion als Request-Parameter mit dem Namen *category* übergeben werden.

@Produces({ APPLICATION_JSON })

Der Rückgabewert der Methode *getProductsOfCategory* soll JSON-codiert als Response ausgeliefert werden.

Exception-Handling

```
@Singleton
@Path("/shop")
public class Shop implements ExceptionMapper<Throwable> {

    ...

    @Override
    public Response toResponse(Throwable exception) {
        exception.printStackTrace();
        System.out.println("REST-Exception: " + exception.getMessage());
        return Response.status(500).entity(exception.getMessage())
            .type("text/plain").build();
    }
}
```


Exception-Handling

- x Wie soll sich Jersey verhalten, wenn es bei der Ausführung einer REST-Funktion innerhalb der Java-Methode zu einer Exception kommt?
- x Dies kann über einen Exception-Mapper für die gesamte Klasse festgelegt werden.
 - ⇒ Die Klasse erbt von der Schnittstelle *ExceptionHandler<Throwable>*
 - ⇒ Die in dieser Schnittstelle enthaltene Methode *toResponse* muss geeignet überschrieben werden.
 - ⇒ In dem Beispiel der Klasse *Shop* wird bei Exceptions eine HTTP-Response mit einem Statuscode 500 (internal error) produziert. Der Rumpf der Methode besteht aus einem Text (Mime-Type: text/plain) mit dem Message-String der Exception.

Anmerkungen

In Jersey gibt es zahlreiche Möglichkeiten zur Implementierung von REST-APIs.

x Request

⇒ Bei der Methode *getProductsOfCategory* werden die Methodenparameter der REST-Funktion als Request-Parameter übergeben. Alternativen:

- x* die Methodenparameter werden als Teilstücke des URL-Pfads übergeben
- x* die Methodenparameter werden als JSON-Text im Nutzdatenteil übergeben

x Response

- ⇒ Die Methode *getProductsOfCategory* erzeugt eine Response vom Typ JSON. Bei der Serialisierung der Rückgabewerte kommt automatisch Jackson zum Einsatz.
- ⇒ Alternativ könnte man die Rückgabewerte beispielsweise auch im XML-Format serialisieren (JAXB).

6.4 AJAX

Javascript als Client für REST-Funktionen

- x Zu einer REST-API kann man unterschiedliche Client-Programme implementieren. Weder bei der Implementierung der REST-API noch bei der Implementierung der REST-Clients ist man auf eine bestimmte Programmiersprache festgelegt.
- x Eine Möglichkeit eine REST-API zu benutzen besteht darin, aus dem Browser heraus mit Hilfe von Javascript auf die REST-API zuzugreifen.
- x Javascript-Code ist entweder Bestandteil einer HTML-Seite oder wird von dieser Seite aus verlinkt. Beim Laden der HTML-Seite gelangt der Javascript-Code in den Browser.
- x Während eine HTML-Seite im Browser angezeigt wird, kommt es in der Seite und in Teilkomponenten der Seite zu verschiedenen Ereignissen (Events):
 - ⇒ Die Seite wird geladen (onload)
 - ⇒ Die Komponente ändert die Größe (onresize)
 - ⇒ Die Komponente wird angeklickt (onclick)
 - ⇒ Der Mauszeiger bewegt sich über der Komponente (onmouseover)
 - ⇒ ...

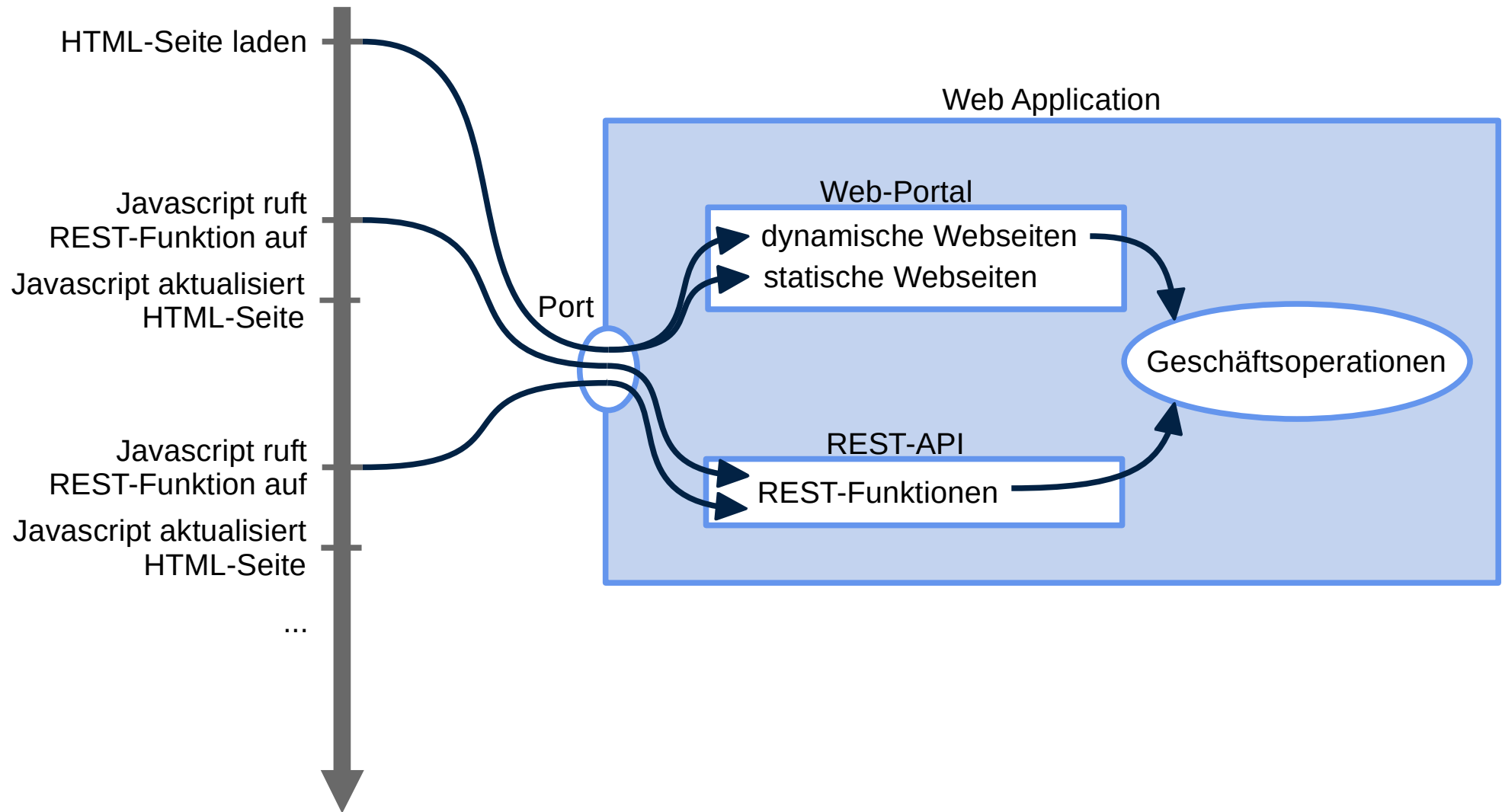
Javascript als Client für REST-Funktionen

- x In dem HTML-Code kann angegeben werden, dass es bei solchen Ereignissen zu Aufrufen von Javascript-Funktionen kommen soll.
- x Eine solche Javascript-Funktion kann dazu verwendet werden, eine REST-Funktion von der Web-Application aufzurufen, von der die HTML-Seite heruntergeladen wurde.
 - ⇒ Die Ausführung der REST-Funktion kann Änderungen bei den Geschäftsdaten bewirken.
 - ⇒ Bei der Ausführung der REST-Funktion können über die Response Daten zurück an den aufrufenden Client (die Javascript-Funktion) übergeben werden.
- x Die Javascript-Funktion kann die Daten, die sie durch den Aufruf der REST-Funktion erhalten hat deserialisieren (parsen) und in geeigneter Weise anzeigen. Dazu gibt es mehrere Möglichkeiten.
 - ⇒ das Ändern des Inhalts der Webseite (Zugriff auf das DOM)
 - ⇒ Popups-Fenster (alerts)
 - ⇒ das Weiterleiten auf andere Seiten

AJAX

- x AJAX: Asynchronous Javascript and XML
- x Beim klassischen Ansatz muss eine neue Seite geladen werden, um den Anwender neue Informationen anzuzeigen ("synchron"). Davon hebt sich das AJAX-Konzept ab.
- x Grundlegender Gedanke:
 - ⇒ Die Inhalte der Seite werden mit Hilfe von Javascript-Funktionen, die REST-Funktionen aufrufen, aktualisiert.
 - ⇒ Benutzerinteraktionen und andere Events führen nicht zum Neuladen einer Seite, sondern zu Änderungen in der Seite.
- x Das X in AJAX steht dafür, dass die Daten bei der Kommunikation mit der REST-API im XML-Format serialisiert werden. Zur Serialisierung wird sehr häufig das JSON-Format anstelle von XML verwendet. Man spricht auch dann von AJAX.

AJAX



AJAX

- x Bei einer AJAX-basierten Anwendung benötigt man nicht unbedingt dynamische Webinhalte (serverseitig dynamisch erzeugte Webseiten).
- x Eine AJAX-basierte Anwendung kann auch nur aus einer einzigen Webseite bestehen. Der Anwender verlässt dann diese eine Seite nicht.
 - ⇒ Single Page Application

HTML, Javascript

- x In HTML-Elemente können Javascript-Funktionen eingefügt werden. Die Javascript-Aufrufe sollen immer dann aufgerufen werden, wenn ein bestimmtes Ereignis eintritt. Beispiele:

```
<body onload="f(x)"> ...
```

⇒ Die Javascript-Funktion f(x) wird aufgerufen, sobald die HTML-Seite geladen wird (onload).

```
<form action="javascript:f(x)"> ...
```

⇒ Die Javascript-Funktion f(x) wird aufgerufen, sobald das Formular abgeschickt wird (action).

- x Möchte man in der Javascript-Funktion ein ganz bestimmtes HTML-Element verändern, so kann man das HTML-Element mit einer ID versehen.

```
<p id="message-element"> ...
```

HTML, Javascript

- x In Javascript können HTTP-Requests mit dem Befehl *fetch* ausgeführt werden.

```
fetch("rest/x/y").then(response => {  
  response.text().then(text => {  
    if (response.ok) {  
      document.getElementById("message-element").innerHTML = text;  
    } else {  
      alert("Operation nicht erfolgreich");  
    }  
  });  
});
```

Erläuterungen

- x Es wird ein HTTP-Request mit dem URL *rest/x/y* ausgeführt.
- x Sobald die Response (Variable: *response*) vorliegt, wird auf den Rumpf der Response zugegriffen. Die Variable *text* erhält den Rumpf der Response in Form eines Strings.
- x Mit den beiden Variablen *response* und *text* wird anschließend eine geeignete Information an den Anwender weitergeleitet.
- x In diesem Beispielprogramm wird in dem if-Konstrukt geprüft, ob der Request erfolgreich war, ob also die Response den HTML-Statuscode OK (200) hat.
 - ⇒ Wenn der Request erfolgreich war, so wird das HTML-Element mit der ID *message-element* verändert. Der Rumpf dieses Elements erhält als Inhalt den Wert der Variable *text*.
 - ⇒ Wenn der Request nicht erfolgreich war, so erscheint auf der Webseite ein Dialogfenster mit der Meldung "Operation nicht erfolgreich".

Anmerkungen

- x Das Beispiel zeigt nur eine mögliche Implementierung für den Aufruf einer REST-Funktion mit Javascript und die anschließende Verarbeitung der Response. Es gibt dazu zahlreiche Alternativen.
- x Beim Aufruf des Requests können Request-Parameter übergeben werden, die aus unterschiedlichen Quellen stammen können. Für die Kodierung der Request-Parameter gibt es unterschiedliche Möglichkeiten.
- x Die Response kann unterschiedliche Datenformate enthalten. Gegebenenfalls müssen die Daten in der Response dekodiert werden. Handelt es sich in der Response beispielsweise um JSON, so werden die Daten mit *JSON.parse* dekodiert.
- x Man kann mit Javascript auf ein einzelnes HTML-Element zugreifen, das man mit einer bestimmten ID versehen hat. Man kann aber auch auf mehrere HTML-Elemente zugreifen und man muss nicht unbedingt mit einer ID arbeiten. Javascript enthält verschiedene Funktionen, um auf einzelne oder mehrere HTML-Elemente einer HTML-Seite zuzugreifen.
- x Oft wird bei der Programmierung von AJAX-Anwendungen die Javascript-Library jQuery eingesetzt. jQuery enthält zahlreiche Funktionen um die Verarbeitung von AJAX-Operationen zu vereinfachen.